Departamento de Ingeniería Telemática

TRABAJO FIN DE GRADO

DESARROLLO E IMPLEMENTACIÓN DEL EXTENSIBLE SERVICE SUBSCRIPTION PROTOCOL

Autor: Jorge Peláez

Tutor: Manuel Urueña

Leganés, junio de 2017

Título: DESARROLLO E IMPLEMENTACIÓN DEL PROTOCOLO XSSP

Autor: Jorge Peláez

Director: Dr. Manuel Urueña

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1. INTRODUCCIÓN

1.1. MOTIVACIÓN

El número de dispositivos que se conectan a Internet aumenta exponencialmente cada año. Además del número de dispositivos, también aumentan enormemente las funcionalidades que ofrecen y la complejidad de su funcionamiento. Un gran problema es que la mayor parte de los usuarios no tienen los conocimientos técnicos para configurar correctamente el funcionamiento de las aplicaciones, especialmente en escenarios dinámicos, por ejemplo, cuando un dispositivo móvil se conecta a una red por primera vez.

Uno de los casos en el que los usuarios tienen que editar manualmente la configuración de sus equipos, es al hacer uso de servicios que se acceden a través de una red, ya sea una red local o Internet, para especificar su localización dentro de dicha red. Tradicionalmente, el usuario debería conocer la existencia del servicio y su localización en la red para hacer uso de él. Los protocolos de descubrimiento de servicios simplifican o eliminan la necesidad de preconfigurar el acceso a los servicios de red.

Existen numerosas protocolos para realizar automáticamente el descubrimiento de servicios, pero los más utilizados son los estándares *Universal Plug and Play* [1] (UPnP), Bonjour [2] y *Service Location Protocol* [3] (SLP).

El *eXtensible Service Discovery Framework* (XSDF) nace como una mejora de SLP que ha sido diseñada para facilitar el descubrimiento de servicios complejos y el reparto de carga en redes de todos los tamaños.

XSDF está compuesto por una serie de protocolos independientes entre sí: el *eXtensible Service Location Protocol* (XSLP), el *eXtensible Service Register Protocol* (XSRP), el *eXtensible Service Subscription Protocol* (XSSP) y el *eXtensible Service Transfer Protocol* (XSTP). En el momento del comienzo de este proyecto, XSLP [4] y XSRP [5] han sido implementados por Trabajos Fin de Grado anteriores.

XSSP permite la suscripción de un usuario a determinados servicios, recibiendo una notificación cada vez que un servicio que cumple con sus requisitos es creado, modificado o eliminado. De este modo, el usuario siempre tendrá la información actualizada sin necesidad de solicitarla periódicamente. Mediante este proyecto, se desarrollará una implementación de un cliente y un servidor XSSP basado en la especificación existente [6].

1.2. OBJETIVOS

Este proyecto se basa en los *Internet Drafts* de XSDF [7] publicados por Manuel Urueña y David Larrabeiti, por lo que se cuenta a priori con la base teórica necesaria para su desarrollo. Dado que los requisitos de la aplicación están perfectamente definidos, se empleará un modelo de desarrollo en cascada en el que cada fase del proyecto existe individualmente sin afectar a las fases anteriores.

Los principales objetivos del proyecto son:

- El estudio en profundidad de la especificación de XSDF y en concreto del *eXtensible Service Subscription Protocol* (XSSP) para realizar una implementación en Java de un cliente y un servidor XSSP que cumplan todas las funciones descritas en los documentos.

- La realización de pruebas de funcionamiento de todos los protocolos de XSDF que ya han sido implementados, para observar su comportamiento conjunto y garantizar un buen funcionamiento entre ellos.
- El desarrollo de la documentación necesaria para la comprensión detallada del trabajo realizado y del funcionamiento de los elementos desarrollados. Esta documentación está formada por esta memoria, y por archivos los HTML generados mediante la herramienta Javadoc que documentan el código.

1.3. MARCO REGULADOR

Todo el proyecto está basado en los *Internet Drafts* de XSDF [7] [8] [9] [10] [11] [6] [12] elaborados por Manuel Urueña y David Larrabeiti, y publicados por el IETF, y se realiza con el consentimiento y supervisión de Manuel Urueña como tutor de este Trabajo Fin de Grado.

Todo el código se distribuye bajo los términos de una Licencia Pública de GNU [13], que permite la libertad de uso, modificación y distribución del mismo.

1.4. IMPACTO SOCIO-ECONÓMICO

La implantación de XSDF en entornos domésticos y corporativos ofrece múltiples ventajas para los usuarios de estas redes.

XSDF permite el descubrimiento automático de los servicios que formen parte de cualquier red, sin necesidad de intervención por parte del usuario. Para éste, no habría diferencia entre emplear un servicio conectado localmente a su dispositivo, o emplear un servicio en red. Gracias a esta característica, un usuario con poco o ningún conocimiento técnico es capaz de acceder a los servicios disponibles. Esto potenciará la aparición y el uso de nuevos servicios de red que aumentarán la presencia de la tecnología en la vida cotidiana.

En redes corporativas, XSDF ofrece un ahorro importante en comparación con otros protocolos de descubrimiento de servicios. Al automatizar el descubrimiento de servicios, no es necesario que un departamento técnico se haga cargo de la configuración estática de los servicios en cada dispositivo, pudiendo añadir o eliminar servicios de forma dinámica y sin costes de despliegue adicionales. Además, se puede garantizar la alta disponibilidad de los servicios ya que, si uno falla, otro puede comenzar a ofrecer las mismas funcionalidades sin que el usuario note la diferencia.

Dado que XSDF es escalable a Internet, las empresas podrán ofrecer servicios a usuarios en cualquier lugar del mundo de forma dinámica y eficiente. Si un usuario quiere saber los servicios que ofrece una determinada empresa únicamente deberá realizar consultas sin necesidad de contactar con los administradores de la red.

Concretamente, XSSP supondrá un ahorro para las empresas, ya que reduce el tráfico necesario para que un usuario posea la información actualizada de un servicio o un grupo de servicios. En caso de no existir XSSP, para que un usuario pudiera tener la información actualizada, debería solicitar su información constantemente, produciendo una gran cantidad de tráfico con información idéntico.

Además, mediante XSSP es posible desplegar servicios críticos con alta disponibilidad en los agentes de directorio, ya que les permite sincronizarse para compartir información idéntica.

1.5. ESTRUCTURA DE LA MEMORIA

Esta memoria ha sido redactada para ser leída de forma secuencial, es decir, para comprender correctamente un capítulo es recomendable haber leído los anteriores. La memoria se divide en nueve capítulos y un anexo con todos los acrónimos empleados para facilitar su lectura:

1. **Introducción**: En el capítulo de introducción se exponen principalmente las motivaciones y los objetivos que se pretenden alcanzar mediante la redacción de la memoria. Además, se analizan el marco regulador que afecta al proyecto, y su impacto socio-económico.
2. **Estado del arte**: En este capítulo se realiza un análisis de los principales protocolos de descubrimiento de servicios similares a XSDF para obtener un mayor conocimiento del problema y para lograr una mejor funcionalidad de XSDF.
3. ***eXtensible Service Discovery Framework* (XSDF)**: En este capítulo se exponen de forma teórica las principales características del entorno de descubrimiento de servicios XSDF, realizando un breve análisis de todos los agentes y protocolos que lo conforman.
4. ***eXtensible Service Subscription Protocol* (XSSP)**: En este capítulo se expone en detalle el funcionamiento de los clientes y servidores XSSP, así como la descripción de los mensajes y elementos involucrados en el protocolo, ya sean o no exclusivos de XSSP.
5. **Diseño de un cliente-servidor XSSP**: En este capítulo se explican detalladamente las funcionalidades que poseía la implementación de XSDF recibida al inicio de este proyecto. También se realiza un análisis de los principales casos de uso del protocolo XSSP.
6. **Implementación de un cliente-servidor XSSP**: En este capítulo se detalla la implementación que se ha realizado de un cliente y un servidor XSSP, así como los cambios realizados para integrarlos en los agentes XSDF ya existentes.
7. **Evaluación y pruebas**: En este capítulo se detallan todas las pruebas realizadas al sistema para comprobar su buen funcionamiento y la inexistencia de casos no contemplados durante el diseño.
8. **Planificación y presupuesto**: En este capítulo se detalla la planificación que se realizó al comenzar el proyecto, y un presupuesto de realización teniendo en cuenta todos los costes derivados del proyecto.
9. **Conclusiones y trabajos futuros**: En este capítulo se comentan las conclusiones obtenidas a partir de la realización del proyecto, y se describen los trabajos necesarios para completar el entorno de XSDF.

Capítulo 2. ESTADO DEL ARTE

2.1. PROTOCOLOS DE DESCUBRIMEINTO DE SERVICIOS

En la última década, el número de dispositivos conectados a Internet ha aumentado exponencialmente, y se prevé que aumente aún más durante los próximos años. El abaratamiento de la tecnología y un profundo cambio en la sociedad están propiciando que exista una mayor diversidad de terminales para el usuario, y una mayor oferta de servicios en red. El problema de esta situación es que, en muchas ocasiones, es necesaria una configuración más o menos compleja para que un terminal (como un ordenador o un móvil, o dispositivos de reciente creación como relojes inteligentes o incluso coches) pueda hacer uso de unos servicios determinados (impresoras, distribución multimedia, o incluso dispositivos IoT que pueden encontrarse en cada vez más hogares).

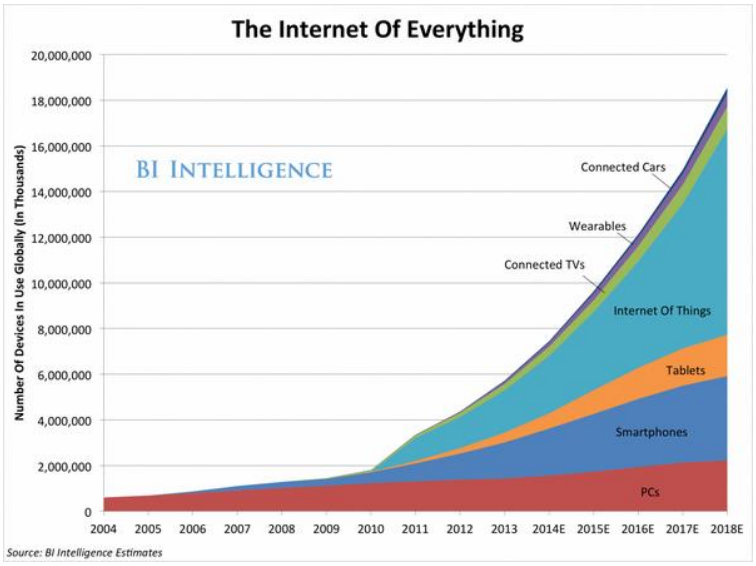

Ilustración 1. Evolución dispositivos conectados [14]

Esta necesidad de configuración por parte de los usuarios, que normalmente carecen de los conocimientos técnicos necesarios para realizarla, es la que motiva el desarrollo de Protocolos de Descubrimiento de Servicios [15] (conocidos como SDPs por las siglas en inglés de *Service Discovery Protocols*). Mediante el uso de SDPs, se automatiza casi totalmente el proceso de localizar los servicios de red a los que se tiene acceso, haciendo transparente al usuario todo el proceso de encontrar y seleccionar el servicio que se ajusta a sus necesidades.

Generalmente en un SDP se reconocen dos entidades:

- El usuario, que mediante un dispositivo desea acceder a un servicio de red.
- El proveedor del servicio, generalmente un servidor que gestiona uno o mas servicios.

Existen múltiples protocolos desarrollados por diferentes organizaciones que permiten el descubrimiento automático de servicios. A grandes rasgos, se pueden clasificar en dos grandes grupos:

- Distribuidos: En un SDP distribuido, los usuarios descubren directamente a los servicios disponibles en la red, normalmente utilizando consultas *multcast*.
- Centralizado: En un SDP centralizado, se incluye una tercera entidad denominada directorio de servicios. El directorio es una entidad intermedia entre el usuario y los servicios, que cuenta con una caché de servicios para reducir el tráfico *multicast* en la red.

En los siguientes apartados se describirán los protocolos de descubrimiento de servicios más empleados en la actualidad: SSDP, perteneciente a UPnP, desarrollado principalmente por Microsoft; Bonjour, desarrollado por Apple; y SLP, desarrollado por IETF.

2.2. UNIVERSAL PLUG AND PLAY (UPNP)

Universal Plug and Play (UPnP) [1] es una tecnología que permite tanto descubrir los servicios de una red, como establecer posteriormente comunicación con los mismos. Está diseñado para proporcionar, de un manera flexible y sencilla, conectividad a redes no gestionadas de pequeño tamaño, corporativas o incluso Internet.

La Arquitectura de Dispositivos UPnP (UDA, por las siglas en inglés de *UPnP Device Architecture*) está diseñada para soportar el descubrimiento de servicios automático y sin configuración previa. Esto se traduce en que un dispositivo puede unirse a una red, conseguir automáticamente una dirección IP, y conocer la presencia y características de otros dispositivos UPnP presentes en la red.

<table>
<tr><td colspan="4">Fabricante UPnP</td></tr>
<tr><td colspan="4">Foro UPnP</td></tr>
<tr><td colspan="4">Arquitectura de dispositivo UPnP</td></tr>
<tr><td rowspan="2">SSDP</td><td rowspan="2">Eventos multicast</td><td>SOAP</td><td>GENA</td></tr>
<tr><td colspan="2">HTTP</td></tr>
<tr><td colspan="2">UDP</td><td colspan="2">TCP</td></tr>
<tr><td colspan="4">IP</td></tr>
</table>

Ilustración 2. Pila de protocolos UPnP [1]

Tras obtener una dirección IP, el siguiente paso para un dispositivo es el descubrimiento de otros dispositivos que se encuentran en la misma red. Cuando se añade un dispositivo a la red, el protocolo de descubrimiento de UPnP, *Simple Service Discovery Protocol* (SSDP), permite al dispositivo anunciar sus propios servicios a los puntos de control de la red. Asimismo, cuando se añade un punto de control a la red, UPnP le permite buscar dispositivos de interés en la red. En ambos casos, la información intercambiada será su tipo de dispositivo, su identificador e información adicional, así como otros parámetros opcionales. SSDP utiliza como protocolo para transportar sus mensajes HTTP/1.1 sobre UDP.

Cuando un dispositivo sabe que acaba de ser añadido a una red, debe enviar mediante *multicast* mensajes `NOTIFY ssdp:alive` anunciándose a sí mismo y a sus servicios. Cualquier punto

de control puede escuchar en esa dirección *multicast* para ser notificado de los nuevos servicios. Cada mensaje debe contener los siguientes componentes:

- El tipo de notificación, indicado en el campo NT (*Notification Type*) de la cabecera.
- Un identificador para el anuncio, indicado en el campo USN (*Unique Service Name*) de la cabecera.
- Una URL para obtener más información sobre el dispositivo, indicada en el campo LOCATION de la cabecera.
- La duración de validez del anuncio, indicada en el campo CACHE-CONTROL de la cabecera.

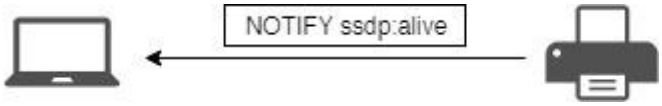

Ilustración 3. SSDP: Anuncio de servicios

Cuando se añade a la red un nuevo punto de control, se permite que envíe mediante *multicast* un mensaje M-SEARCH para descubrir dispositivos y servicios. Todos los dispositivos deben escuchar este tipo de mensaje y responderlos.

En caso de que un dispositivo que haya recibido un mensaje M-SEARCH coincida con los criterios especificados en la búsqueda, debe responder con un mensaje RESPONSE.

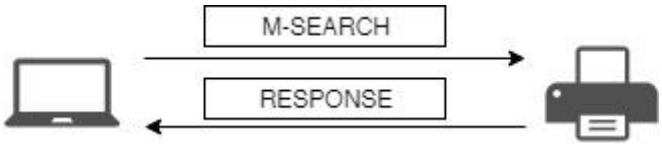

Ilustración 4. SSDP: Descubrimiento de servicios

Cuando un dispositivo se elimina de la red, debe, en caso de ser posible, enviar mensajes `NOTIFY ssdp:byebye` notificando que sus anuncios previos dejan de ser válidos y que sus servicios dejarán de estar disponibles. Cada mensaje debe contener los siguientes componentes:

- El tipo de notificación, indicado en el campo NT de la cabecera.
- Un identificador para el anuncio, indicado en el campo USN de la cabecera.

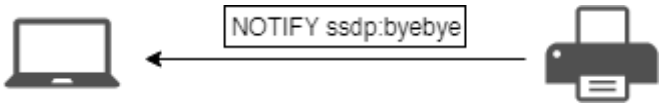

Ilustración 5. SSDP: Eliminación de servicios

2.3. BONJOUR

Bonjour [2] es una tecnología de código abierto desarrollada por Apple que permite conectar automáticamente dispositivos a una red, permitiéndoles interactuar ofreciendo y recibiendo servicios sin necesidad de configuración estática por parte del usuario. Bonjour está basado en un conjunto de estándares desarrollados por el IETF como *multicast DNS* (mDNS) y *DNS-Service Discovery* (DNS-SD).

Con Bonjour es posible conectar un dispositivo a una red existente, o crear una nueva red. Bonjour asigna automáticamente una dirección IP a cada miembro de la red y permite el uso de sus servicios a los demás miembros. Bonjour está diseñado para funcionar en redes pequeñas, como redes *ad-hoc*, o redes locales (*Local Area Networks*, LANs).

El primer paso cuando un dispositivo se conecta a una red, es asignarle una dirección IP para que pueda comenzar a transmitir mensajes. Si hay desplegado un servidor DHCP, se utilizará la configuración que provea DHCP. Si no, el dispositivo se asignará a sí mismo una dirección de tipo *link-local* (el rango de direcciones *link-local* está establecido por la IANA, y se compone de las direcciones 169.254.xxx.xxx) y mandará un mensaje ARP a la red para comprobar que esa dirección no está en uso. Si otro dispositivo emplea la misma dirección, el nuevo dispositivo cambiará su dirección hasta hallar una libre.

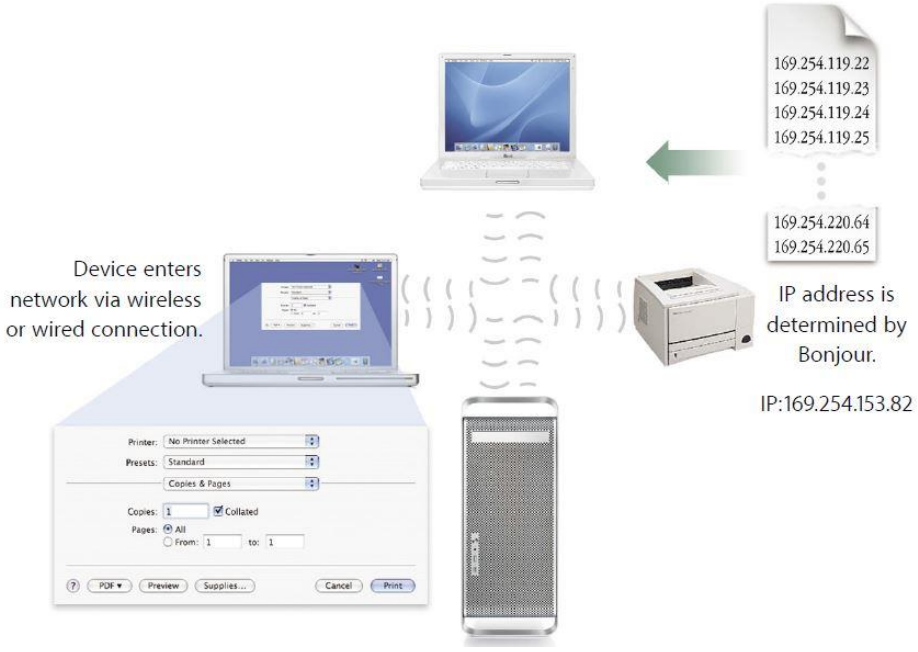

Ilustración 6. Bonjour: Asignación automática de dirección IP [2]

Una vez el dispositivo forma parte de la red, necesita un mecanismo para descubrir qué servicios ofrecen otros dispositivos, y para anunciar sus propios servicios.

Para compartir un servicio, el dispositivo primero debe asignarle un nombre DNS. Dado que en las redes en las que opera Bonjour normalmente no existen servidores DNS, emplea una variante denominada *Multicast DNS - Service Discovery* (mDNS-SD). Una notificación de mDNS-SD contiene el tipo de servicio, el nombre del servicio, su dirección IP y puertos de comunicación, e información adicional. Dado que la notificación se envía por *multicast*, todos los dispositivos de la red la reciben y guardan la información.

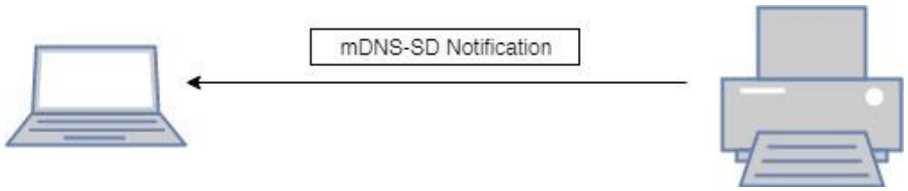

Ilustración 7. Bonjour: Anuncio de servicios

Cuando un dispositivo nuevo quiere recibir la información sobre los servicios de un determinado tipo existentes en la red, realiza una consulta mDNS-SD que será respondida por los servicios de dicho tipo.

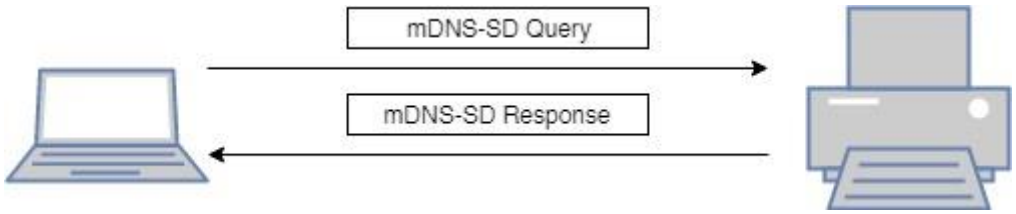

Ilustración 8. Bonjour: Descubrimiento de servicios

2.4. SERVICE LOCATION PROTOCOL (SLP)

Service Location Protocol (SLP) [16], es un protocolo de descubrimiento de servicios desarrollado por el IETF. En 1997 se publicó SLPv1, y en 1999 se publicó SLPv2, que corregía inconsistencias de la primera versión. En la actualidad, SLPv2 es la última versión oficial.

SLP proporciona un sistema de descubrimiento de servicios flexible y ligero. Además, SLP proporciona mecanismos de escalabilidad, por lo que puede ser empleado tanto en redes pequeñas como en grandes redes corporativas. A diferencia de UPnP, SLP solo permite localizar un servicio, posteriormente deberá utilizarse un protocolo más indicado para interactuar con el servicio seleccionado.

SLP [3] establece un escenario en el que las aplicaciones del cliente se denominan Agentes de Usuario (UA, *User Agent*), y los servicios son anunciados por Agentes de Servicio (SA, *Service Agent*). Una tercera entidad opcional llamada Agente de Directorio (DA, *Directory Agent*) hace de intermediaria entre los UAs y los SAs para proporcionar escalabilidad en redes grandes.

En SLP los servicios se identifican mediante URLs [17] que también indican su localización. Una URL puede tener la forma de:

service:<srvtype>://<addrspec>

Todas las URL que identifican un servicio en SLP comienzan con "service:".

Después se especifica el tipo o tipos del servicio sustituyendo el campo <srvtype>. En caso de existir más de uno, se usa la sintaxis:

service:<abstract-type>:<concrete-type>

Finalmente, sustituyendo al campo <addrspec> debe indicarse el nombre del host o la dirección IP del servicio, seguido del puerto.

El anuncio de un servicio puede ir acompañado de una serie de atributos. Generalmente, cada tipo de servicio tiene asociada una serie de atributos que pueden ser empleados para describir un servicio. Estos atributos pueden ser empleados por los UAs en los mensajes SrvRqst para

seleccionar el servicio apropiado, aunque normalmente le presentan al usuario todos los servicios descubiertos para que este seleccione uno manualmente.

Para descubrir servicios, un UA envía un mensaje *Service Request* (SrvRqst) especificando las características del servicio que la aplicación necesita. Este mensaje será contestado con un *Service Reply* (SrvRply) que especifica la localización de todos los servicios que cumplen con las condiciones. El mensaje SrvRqst puede ser enviado a todos los SA mediante *multicast*. Los SAs que anuncien un servicio que cumpla con lo requerido, responderán con un mensaje SrvRply mediante *unicast* directamente al UA.

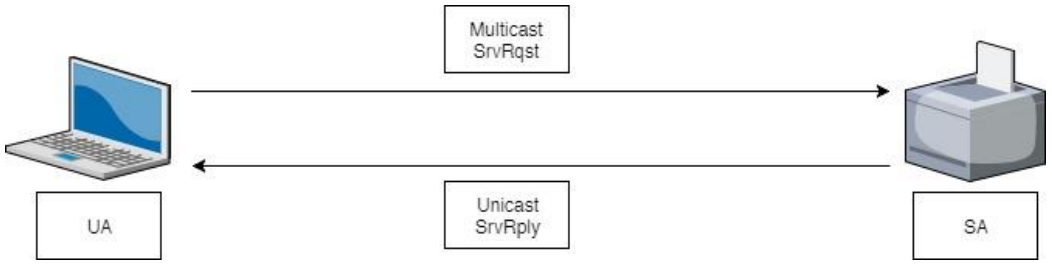

Ilustración 9. SLP: Descubrimiento de servicios mediante multicast

En redes de mayor tamaño se pueden desplegar uno o más DAs. Éstos actúan como una caché centralizada en la que los SAs registran los servicios que anuncian en todos los DAs existentes mediante el envío de mensajes *Service Register* (SrvReg). Los mensajes SrvReg se responden con confirmaciones SrvAck. Estos registros son *soft-state*, es decir, deben ser actualizados periódicamente para no ser eliminados.

Ilustración 10. SLP: Registro de servicios

En este escenario, si los UAs conocen algún DA, envían los SrvRqst al DA en *unicast* para evitar el tráfico *multicast*, cuyo uso no es adecuado en redes grandes.

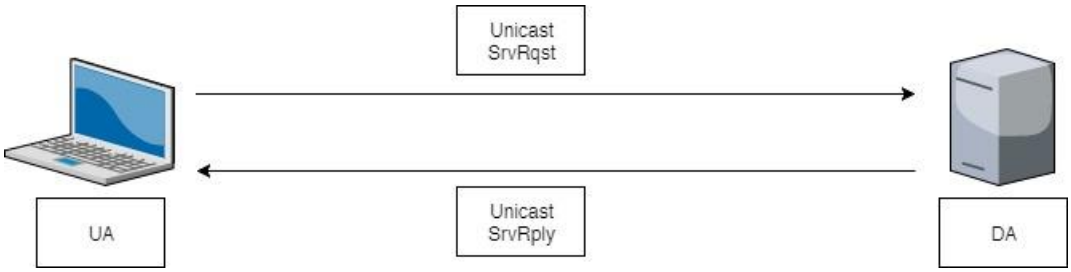

Ilustración 11. SLP descubrimiento de servicios mediante DA

Los UAs y los SAs pueden descubrir a los DAs de dos maneras:

1. Mediante el envío de un mensaje SrvRqst *multicast* solicitando servicios de tipo *Directory Agent*, que son respondidos con un mensaje DAAdvert.
2. Mediante la recepción de un mensaje DAAdvert *multicast*, enviado por los DAs de forma periódica.

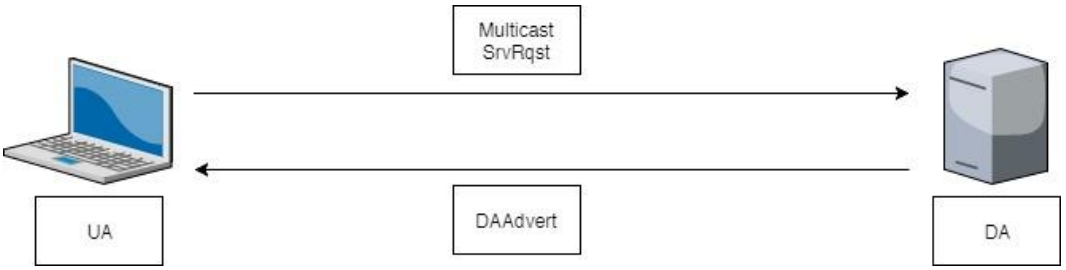

Ilustración 12. SLP: Descubrimiento de DA

Se pueden organizar servicios en grupos denominados ámbitos (*scopes* en inglés), que pueden indicar un grupo administrativo o cualquier otra agrupación lógica. Los DAs y los SAs siempre están asociados a un ámbito. Un UA puede estar asociado a un ámbito para definir a qué servicios podrá acceder, o puede no estar asociado a ninguno y solicitar cualquier servicio de la red.

Hay varias funcionalidades opcionales en SLP. Los DAs deben soportar todos los mensajes que se escoja utilizar en un despliegue de SLP. Los UAs y SAs pueden o no soportar las funcionalidades opcionales. Estas operaciones sirven principalmente para proporcionar un uso interactivo.

Mensaje	Descripción
Service Type Request	Mensaje empleado para descubrir todos los tipos de servicios que hay en la red.
Service Type Reply	Mensaje empleado para responder a la solicitud *Service Type Request*.
Attribute Request	Mensaje empleado para descubrir los atributos de un determinado tipo de servicio, o de un determinado servicio.
Attribute Reply	Mensaje empleado para responder a la solicitud *Attribute Request*.
Service Deregister	Mensaje empleado para eliminar el registro de un servicio, o alguno de los atributos de un registro.
Service Update	Mensaje empleado para actualizar atributos dinámicos de un servicio.
SA Advertisement	Mensaje que, en ausencia de DAs, permite a los SAs anunciarse a los UAs.

Tabla 1. SLP: Mensajes opcionales

Los SA deben aceptar mensajes SrvRqst tanto *unicast* como *multicast,* y además pueden aceptar mensajes *Service Type Request* y *Attribute Request*. Los SAs deben escuchar los mensajes DAAdvert.

En caso de que la red no soporte mensajes *multicast*, pueden enviarse mediante broadcast. Adicionalmente, la localización de los DAs puede ser configurada estáticamente o mediante DHCP.

Capítulo 3. EXTENSIBLE SERVICE DISCOVERY FRAMEWORK (XSDF)

En los siguientes apartados se describe detalladamente el entorno de descubrimiento de servicios XSDF y sus características.

XSDF [7] fue inicialmente propuesto y diseñado por Manuel Urueña y David Larrabeiti, y fue publicado en el IETF como Internet-Draft en 2004.

En los capítulos anteriores se ha tratado la necesidad de un protocolo de descubrimiento de servicios efectivo y eficiente, con el objetivo de simplificar o eliminar la configuración estática de cualquier dispositivo de usuario o servicio.

Para cubrir esta necesidad mejorando el funcionamiento de los protocolos existentes, nace el *eXtensible Service Discovery Framework* (XSDF). XSDF es un entorno de descubrimiento de servicios que opta por el uso de múltiples protocolos sencillos en lugar de un único protocolo complejo. De este modo se logra un desarrollo y despliegue mucho más sencillos, ya que cada protocolo es independiente de los demás. Además, cada agente involucrado necesitaría únicamente el código asociado al protocolo o protocolos que utiliza, haciéndolos mucho más rápidos y reduciendo en gran medida su consumo energético.

XSDF es una evolución del protocolo SLP explicado en el capítulo anterior, combinado con mecanismos de reparto de carga basado en Rserpool [18].

Las principales mejoras de XSDF respecto a SLP son:

- Identificación de los servicios: Los servicios en XSDF se identifican mediante un *Universally Unique Identifier* [19] (UUID) y un tipo, que permiten agrupar servicios de forma jerárquica. En SLP, los servicios se identificaban mediante una URL más una serie de atributos opcionales.
- Escalabilidad: Mientras que SLP fue diseñado para funcionar en redes LAN y redes corporativas, XSDF puede funcionar en redes mucho más grandes o incluso Internet.
- Reparto de carga: XSDF incluye mecanismos de reparto de carga para mejorar la eficiencia de los servicios. Estos mecanismos puede realizarlos el dispositivo del usuario o el servidor según sea necesario.
- Alta disponibilidad: XSDF permite sincronizar varios servidores de directorio para que contengan la misma información, de modo que, si un servidor deja de funcionar, el resto seguirán funcionando.
- Configuración innecesaria: En XSDF, los servidores son considerados servicios para poder ser descubiertos del mismo modo que cualquier otro servicio. De esta forma se elimina la necesidad de establecer mensajes específicos para localizar agentes XSDF.

3.1. AGENTES XSDF

El entorno de descubrimiento de servicios XSDF define cuatro tipos de agentes que pueden existir o no en el despliegue de una red:

- Agente de Usuario (UA por sus siglas en inglés*, User Agent*): Es un proceso que se ejecuta en la máquina del usuario cuya funcionalidad consiste en descubrir los servicios

disponibles en una red y, en caso de haber varios de un mismo tipo, decidir cuál es mejor. Solicitará información de servicio a los SAs o DAs.

- Agente de Servicio (SA por sus siglas en inglés, *Service Agent*): Es un proceso que se ejecuta en un servidor, cuya finalidad consiste en anunciar los servicios proporcionados por el mismo. Un SA es considerado también un servicio. Proporcionará información de servicio a los UAs cuando lo soliciten, o a los DAs cuando estén desplegados.
- Agente de Directorio (DA por sus siglas en inglés, *Directory Agent*): Es un agente opcional, considerado también un servicio, que actúa como repositorio central de información. Proporcionará información de servicio a los UAs cuando lo soliciten, que habrá sido registrada previamente por los SAs.

El objetivo principal del entorno de descubrimiento de servicios XSDF es que los UAs logren obtener la información de los servicios disponibles en una red, de la forma más eficiente y transparente posible. Esta información podrá ser proporcionada directamente por los SAs, con la posibilidad de obtenerla a través de un DA centralizado.

En redes pequeñas, los UAs pueden solicitar la información directamente a los SAs mediante mensajes XSLP *multicast*.

En redes más grandes es necesario emplear un DA. Un DA no genera información de servicios, sino que actúa como repositorio central de la misma. Los SAs actualizan periódicamente la información de servicio en los DAs mediante XSRP. Los UAs obtienen la información de los servicios mediante consultas XSLP *unicast* a los DAs.

Cuando varios DAs mantienen la información de los mismos servicios, todos los DAs deben estar sincronizados para poder ofrecer exactamente la misma información a los UAs. En este caso los DAs utilizan XSTP para obtener inicialmente la información de otros DAS y XSSP para suscribirse a los cambios que se realicen en otros DAs y mantenerse sincronizados.

3.2. PROTOCOLOS XSDF

La arquitectura XSDF está compuesta por múltiples agentes XSDF y protocolos de comunicación. Cada uno de los protocolos se encarga de una parte diferente del proceso de descubrimiento de servicios.

Todos los protocolos de XSDF emplean un modelo cliente-servidor muy simple. Esto permite que los UAs y SAs sean instalados en plataformas de embebidas.

Los mensajes de XSDF podrían ser codificados con cualquier lenguaje estructurado jerárquicamente, como puede ser el caso de XML. Sin embargo, se recomienda el uso del lenguaje *eXtesible Binary Encoding 32* (XBE32) [9] ya que, gracias a estar codificado de forma binaria, es mucho más compacto y fácil de procesar que XML.

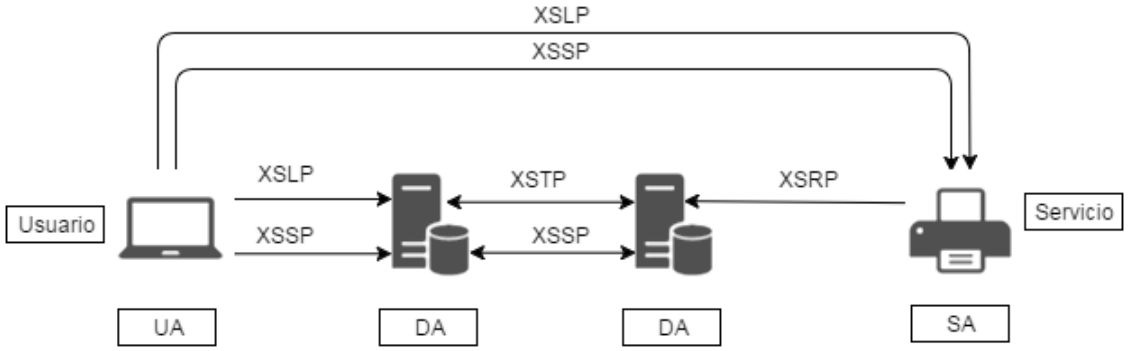

Ilustración 13. Agentes y protocolos XSDF

3.2.1. EXTENSIBLE SERVICE LOCATION PROTOCOL (XSLP)

XSLP [10] es el protocolo empleado por los UAs para obtener información de servicio de los SAs o DAs. Esta información incluye la descripción y la localización del servicio, datos sobre su estado actual para escoger el mejor si hay varios, y el tiempo de validez de esta información de servicio para poder almacenarla en una caché. Mediante este protocolo, un UA puede localizar el mejor servicio sin ningún tipo de configuración estática o intervención por parte del usuario.

Los mensajes de XSLP pueden enviarse en *multicast* a varios SAs (o *broadcast* si el servicio *multicast* no está soportado), o en *unicast* a un DA en caso de que el tamaño de la red haya requerido su despliegue. Las respuestas, tanto de los SAs como de los Das, se enviarán al UA por *unicast*.

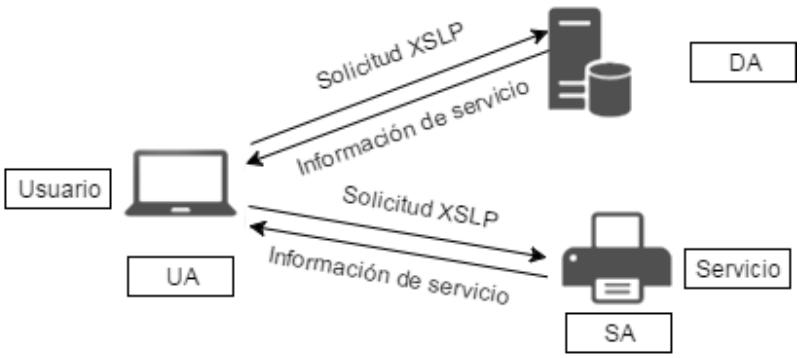

Ilustración 14. Protocolo XSLP

3.2.2. EXTENSIBLE SERVICE REGISTER PROTOCOL (XSRP)

XSRP [11] es el protocolo empleado por los SAs para registrar su información de servicio en uno de los DAs que gestionan el ámbito al que pertenece el servicio. De este modo, los DAs pueden actuar como repositorios centralizados de servicios de un ámbito. La información registrada por los SAs debe actualizarse periódicamente para que no sea eliminada por el DA. Además, cuando un servicio deje de estar disponible, deberá notificarse al DA para que elimine su registro.

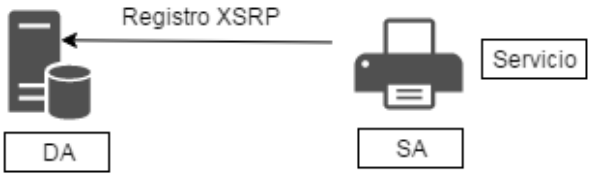

Ilustración 15. Protocolo XSRP

3.2.3. EXTENSIBLE SERVICE SUBSCRIPTION PROTOCOL (XSSP)

XSSP [6] es un protocolo que puede ser empleado por cualquier agente XSDF para suscribirse a los cambios que se realicen en la información de servicio de otro agente XSDF, generalmente un SA o DA, ya que son los que gestionan información de servicios. De este modo, un agente XSDF que necesite tener la información de un servicio siempre actualizada podrá recibir notificaciones cada vez que se produzca un cambio. Sin XSSP, sería necesario enviar solicitudes XSLP periódicamente, se hayan producido cambios o no.

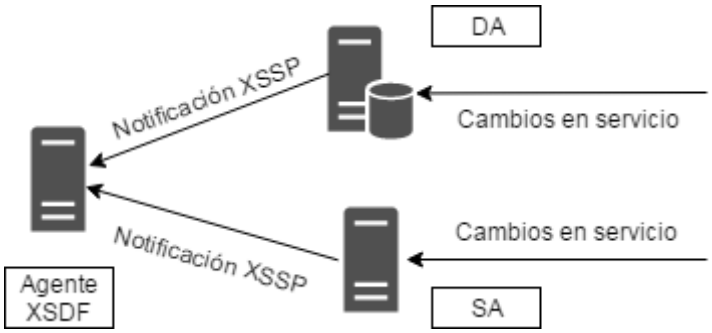

Ilustración 16. Protocolo XSSP

3.2.4. EXTENSIBLE SERVICE TRANSFER PROTOCOL (XSTP)

XSTP [12] es el protocolo empleado por los DAs del mismo reino para descargar toda la información actualizada de los servicios de un ámbito. De este modo, varios DAs pueden ofrecer la misma información, pudiendo repartir las solicitudes XSLP y XSRP recibidas entre ellos. Gracias a esta sincronización se logra escalar el modelo de XSDF a redes de gran tamaño, y además puede utilizarse como mecanismo de alta disponibilidad. Un DA utiliza primero XSTP para obtener todos los servicios gestionados por otro DA, y luego XSSP para suscribirse a los cambios que se realicen en esos servicios.

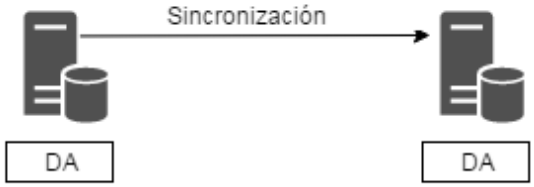

Ilustración 17. Protocolo XSTP

3.3. ELEMENTOS DE XSDF

3.3.1. ÁMBITOS, DOMINIOS Y REINOS

Al igual que SLP, XSDF utiliza una estructura de ámbitos (*scopes* en inglés) para agrupar usuarios y servicios dentro de grupos administrativos. Los usuarios únicamente pueden buscar servicios que se encuentren en su mismo grupo. Además, XSDF emplea el concepto 'dominio' (*domain* en inglés) para permitir descubrimiento de servicios remoto.

Un reino (*realm* en inglés) es la combinación de un dominio, si lo hay, y uno o varios ámbitos. Los agentes XSDF solo pueden pertenecer a un reino, aunque los UAs puedan buscar servicios en otros reinos, es decir, en otros dominios.

Cada organización tiene la posibilidad de definir diferentes ámbitos según sus necesidades. Además, XSDF proporciona tres ámbitos por defecto:

1. DEFAULT: Por defecto, si no se requiere ningún ámbito específico, todos los servicios pertenecen al ámbito DEFAULT y carecen de dominio. La utilización de este ámbito es recomendada para redes no gestionadas.
2. LOCAL: Los servicios que pertenecen al ámbito LOCAL solo son relevantes en el servidor local. Por ejemplo, servicios como SNMP o SSH.
3. PUBLIC: Los servicios que pertenecen al ámbito PUBLIC y pertenecen a un determinado dominio, son servicios públicos que podrán accederse desde fuera del reino.

Con el objetivo de simplificar la implementación de los SAs y UAs, los DAs del mismo ámbito se comportan como si fueran uno solo, por lo que, aunque servicio se puede registrar en un único DA, estará disponible su información en cualquiera de ellos. Cabe destacar que esto solo ocurre dentro de un ámbito, por lo que, si un servicio está disponible en más ámbitos, deberá registrarse en todos los Das responsables de ellos.

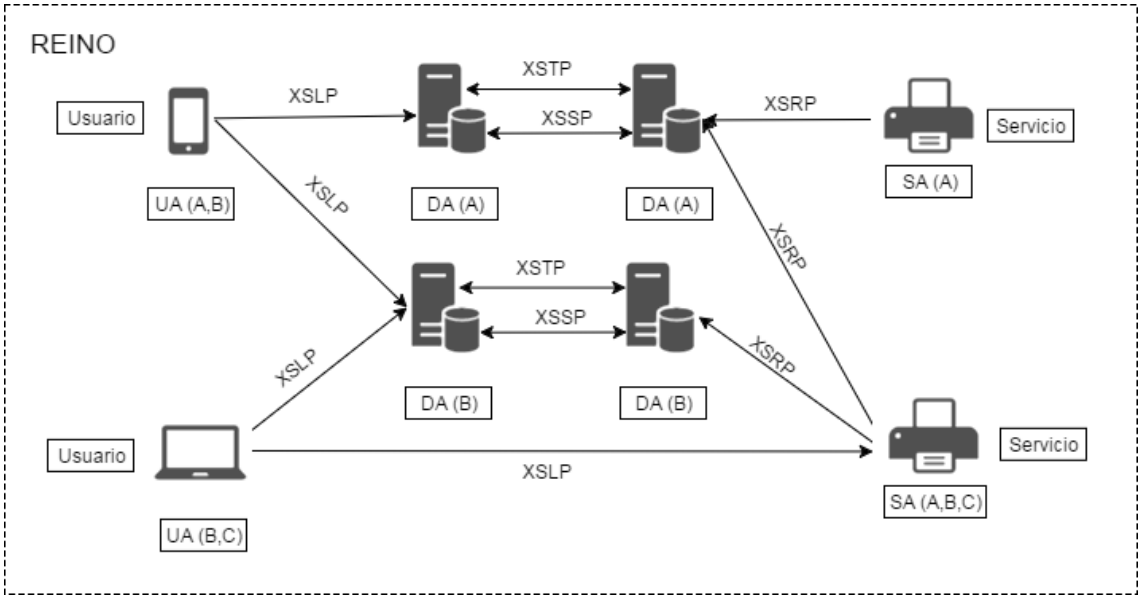

Ilustración 18. Modelo de reinos XSDF

3.3.2. SERVICIOS

En XSDF se combina el modelado de servicios de SLP con el de Rserpool. La información de un determinado servicio está dividida en cuatro partes:

1. Identificador del servicio: Todos los servicios están identificados mediante un UUID único en el dominio al que pertenecen.
2. Estado del servicio: En esta parte se encuentra la información que expirará rápidamente, como el estado de carga del servidor.
3. Información principal: En esta parte se define el tipo del servicio, la política de reparto de carga y otra información relevante del servicio.
4. Información de localización: En esta parte se incluye la información necesaria para que un servicio de red pueda ser localizado, como su dirección IP, los puertos que deben utilizarse y los protocolos disponibles para establecer la comunicación con el mismo.
5. Información adicional: Esta parte no es necesaria para el funcionamiento del descubrimiento de servicios, pero incluye información relevante para el usuario, como una descripción textual del servicio, o la dirección de correo electrónico del administrador del servicio.

En la siguiente figura se muestra un ejemplo de un servicio XSDF. Los detalles de cada elemento pueden encontrarse en la sección 4.1.11 de esta memoria.

```
<service>
    <id> 8e9d7823-8e9d-8e9d-8e9d-fb07ea0c3fb2 </id>
    <serviceState>
        <stateTimestamp> 0 </stateTimestamp>
        <selectState>
            <workload> 0 </workload>
        </selectState>
    </serviceState>
    <serviceMainInfo>
        <serviceType>
            <type> printer </type>
        </serviceType>
        <alias> Old printer </alias>
        <selectInfo>
            <policies> Max Priority (0x0003), Least Used (0x0004) </policies>
            <priority> 5 </priority>
        </selectInfo>
        <serviceOptions>
            <color  ext="true" c="true" e="false" type="boolean"> false </color>
            <duplex ext="true" c="true" e="false" type="boolean"> false </duplex>
        </serviceOptions>
    </serviceMainInfo>
    <serviceNetInfo>
        <inet>
            <ipv4Addrs> 169.254.10.10 (0xA9FE0A0A) </ipv4Addrs>
            <hostname> printer1.local. </hostname>
        </inet>
        <protocol>
            <protoName> acme:printdirect </protoName>
            <transPorts> tcp/9200 (0x000623F0) </transPorts>
        </protocol>
        <protocol>
            <protoName> lpr </protoName>
            <transPorts> tcp/515 (0x00060203) </transPorts>
        </protocol>
    </serviceNetInfo>
    <serviceAddInfo>
        <url> http://printer.local/index.html </url>
        <vendor> Acme Corporation </vendor>
        <vendorURL> http://www.acme.com/index.html </vendorURL>
        <model> Acme Matrix Printer 1000 </model>
        <modelURL> http://www.acme.com/printers/mp1000.html </modelURL>
        <version> 3.2 </version>
        <location> Basement </location>
    </serviceAddInfo>
</service>
```

Ilustración 19. Ejemplo de servicio

3.3.3. CACHÉ DE SERVICIOS

En XSDF, la información de un servicio puede encontrarse en múltiples lugares. Los SAs generan la información, por lo que ningún otro agente XSDF debería modificar esa información. Sin embargo, para mejorar la eficiencia de la red, los SAs pueden registrar esa información en DAs, que actuarán como un repositorio centralizado de servicios.

Al enviar información de servicio, los SAs incluirán un tiempo de validez para esa información. Durante ese tiempo, los DAs podrán utilizar y retransmitir esa información, pero deberán descartarla al finalizar dicho tiempo. Para mejorar aún más la eficiencia de la red, los UAs también pueden guardar información en una caché durante su tiempo de validez, pero no podrán retransmitirla a otros agentes XSDF.

Cuando un DA retransmite la información de un servicio, además del tiempo de validez enviado por el SA, deberán incluir el tiempo de vida que lleva la información sin ser actualizado. De esta manera, un UA que solicite información a un DA podrá utilizarla únicamente durante el tiempo de validez menos el tiempo de vida.

3.3.4. REPARTO DE CARGA

Cuando un UA solicita información mediante XSLP, puede conseguir la información de varios servicios que cumplen sus requisitos. En este caso, podría emplear algún tipo de mecanismo automático que le permita elegir al mejor de ellos automáticamente.

En XSDF se emplea un modelo de selección similar al definido por Rserpool [18]: cada tipo de servicio puede llevar asociada una política de selección, y cada servicio debe proporcionar medidas de lo 'bueno' que es (por ejemplo, en términos de peso, carga, recursos y/o prioridad).

XSDF proporciona varias políticas de selección:

- *Round Robin*: Esta política de selección consiste en que cada vez que se requiera un servicio, se empleará uno diferente. Además, cada servicio puede tener asociado un peso, que modificará las probabilidades de ser usado. Cuanto mayor sea el peso asociado a un servicio, mayor será la probabilidad de que sea escogido.
- Menos usado: Esta política de selección consiste en escoger el servidor con la menor carga. Dado que la carga es un parámetro que cambia contantemente, los agentes XSDF tienen que haberlo solicitarlo recientemente.
- Más recursos: Esta política de selección consiste en escoger el servidor que cuenta con más recursos disponibles. Dado que los recursos disponibles son un parámetro que cambia contantemente, los agentes XSDF tienen que haberlo solicitarlo recientemente.
- Más cercano: Esta política de selección consiste en escoger el servidor al que menos tiempo tarda en responder. En el *framework* de XSDF no se proporcionan las herramientas necesarias para medir la latencia, por lo que el tiempo de respuesta deberá ser obtenido por otros medios (por ejemplo, mediante un *ping*).

Si el servicio escogido por una de las políticas de reparto de carga no tiene recursos disponibles o se encuentra fuera de servicio, se deberá escoger otro.

Además, es posible definir una determinada prioridad para cada servicio. Se debe emplear el servicio de mayor prioridad, y si varios tienen la misma prioridad se empleará su política de selección para decidir.

En XSDF se permite que la selección de servicio sea realizada por el DA, por el UA o por ambos. Es decir, el DA al recibir una petición XSLP de un servicio puede enviar únicamente la información del mejor servicio (el DA decide), la información de varios servicios ordenados de mejor a peor (ambos deciden), o la información de los servicios sin ordenar (el UA decide).

3.4. ESCENARIOS

XSDF puede ser desplegado en multitud de escenarios, desde pequeñas redes no gestionadas hasta redes de grandes empresas o incluso Internet. Dado que cada red tiene sus propias necesidades, cada escenario debería diseñarse de forma individual, pudiendo necesitar todos los agentes y protocolos de XSDF o solo parte de ellos.

3.4.1. REDES NO GESTIONADAS

El primer escenario consiste en una única red de área local que carece de infraestructura de servidores, o posee una muy pequeña. Por ejemplo, un entorno doméstico o SOHO (del inglés *Small Office, Home Office*) con un número reducido de clientes.

En este escenario, el descubrimiento de servicios puede lograrse mediante el uso de UAs y SAs en el mismo ámbito DEFAULT, sin necesidad de desplegar ningún DA. Para encontrar a los SAs, el UA realizará consultas XSLP *multicast*. En caso de encontrar más de un servicio con las características deseadas, serán los UAs los que procedan a su selección.

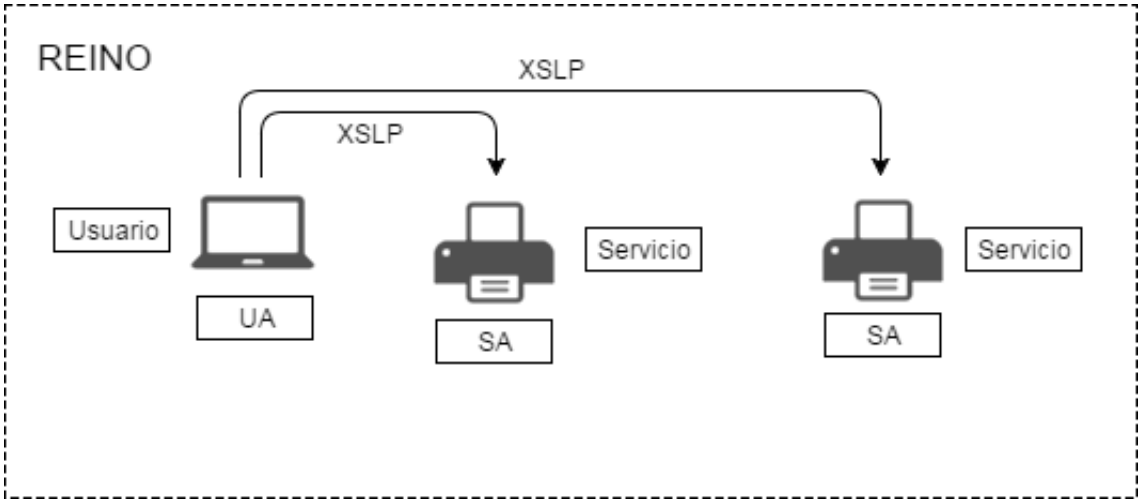

Ilustración 20. Escenario de redes no gestionadas

3.4.2. REDES PEQUEÑAS

El segundo escenario es una red compuesta por varias LANs, donde enviar solicitudes *multicast* sería ineficiente. En este caso se debería desplegar un DA, en el que los SAs registran sus servicios mediante XSRP.

Si el número de usuarios y servicios es reducido, todos los agentes XSDF pueden pertenecer al mismo ámbito, aunque pueden definirse varios. Estos ámbitos están gestionados por uno o varios DAs, por ejemplo, cada DA gestiona un ámbito diferente. Dado que los DAs hacen de

intermediarios entre los UAs y los SAs, los DAs pueden realizar el proceso de selección e incluir en la respuesta los servicios ordenados según la carga del servidor.

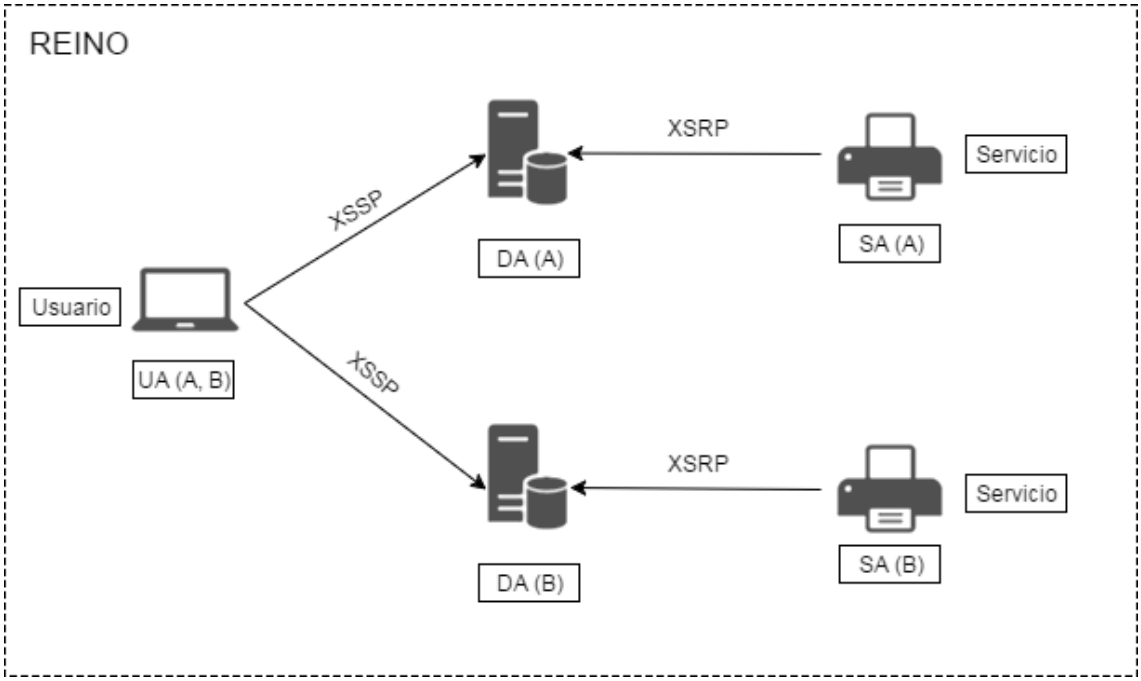

Ilustración 21. Escenario de redes pequeñas

3.4.3. REDES CORPORATIVAS

El tercer escenario es una red corporativa donde existen muchos grupos de usuarios y servicios, y donde pueden existir servicios críticos. En este caso es necesario, además de escalar la arquitectura de XSDF, debe proveer de mecanismos de alta disponibilidad.

En este escenario, los ámbitos que contengan servicios críticos deberían estar gestionados por dos o más DAs sincronizados, para que, en caso de fallo de alguno de ellos, no exista interrupción alguna en el descubrimiento de servicios. Los protocolos XSSP y XSTP permiten que varios DAs de un mismo ámbito compartan la misma información de manera eficiente.

Finalmente, el proceso de selección podrá realizarse tanto en los DAs como en los UAs para reaccionar rápidamente a errores en los servidores.

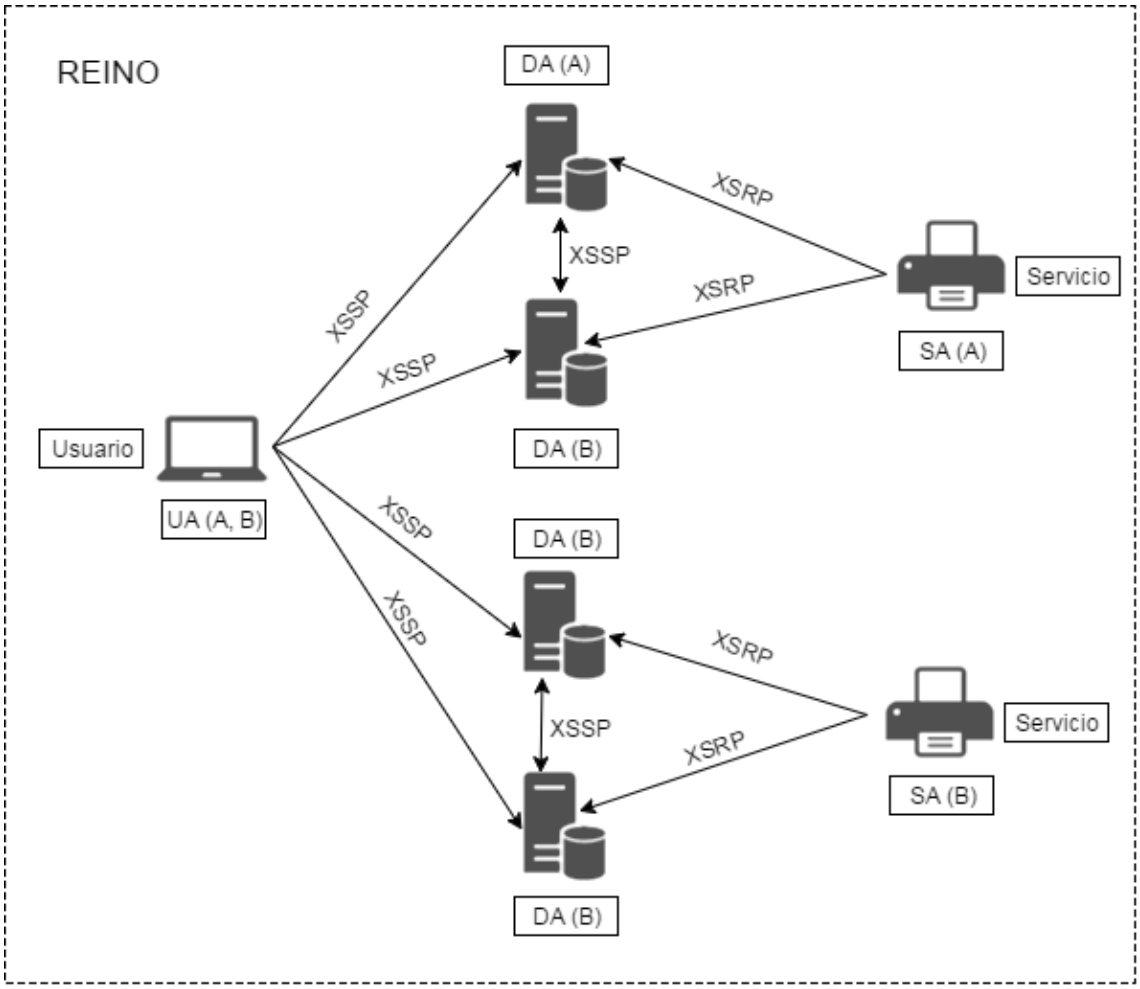

Ilustración 22. Escenario de redes corporativas

3.4.4. INTERNET

Por último, XSDF también podría emplearse para anunciar servicios públicos en Internet. Por ejemplo, un conjunto de servidores web podría emplear XSDF para repartir los clientes de forma eficiente entre ellos.

Este anuncio de servicios públicos se podría llevar a cabo mediante el despliegue de DAs que gestionan un conjunto de servicios del ámbito PUBLIC. A su vez, esos DAs se publican en registros DNS de la organización que quiere ofrecer dichos servicios públicos.

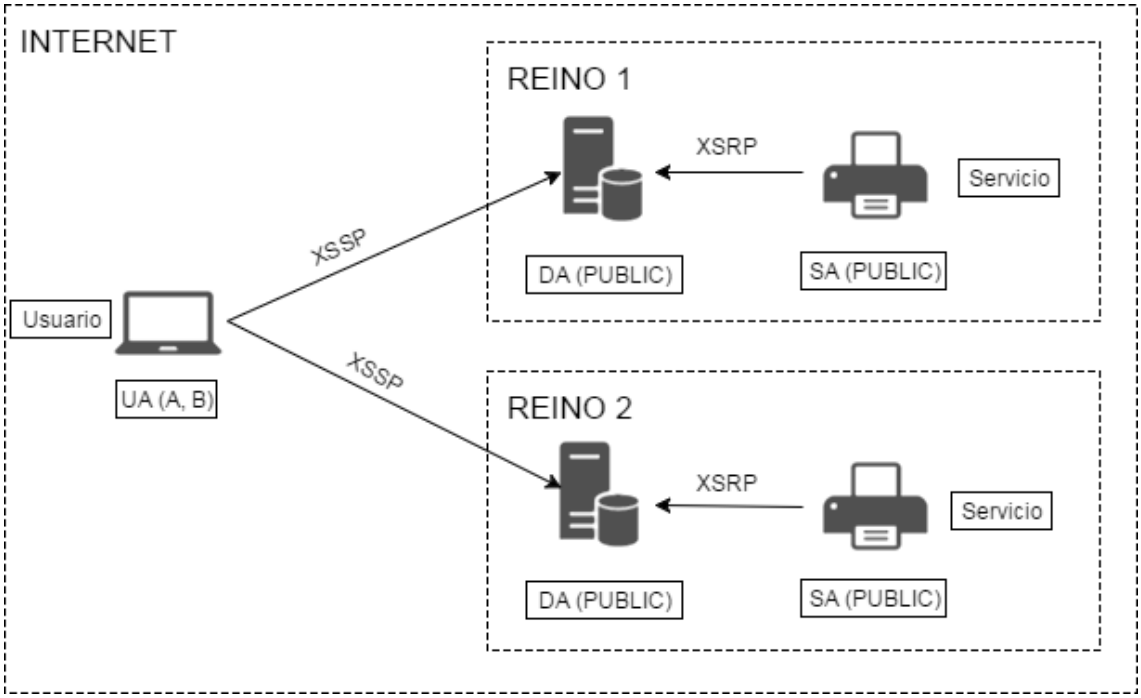

Ilustración 23. Escenario de Internet

Capítulo 4. EXTENSIBLE SERVICE SUBSCRIPTION PROTOCOL (XSSP)

En los siguientes apartados se describe detalladamente el funcionamiento del protocolo XSSP como parte del entorno de descubrimiento de servicios XSDF.

Este capítulo está escrito de acuerdo al diseño original de XSSP, tomando como fuente el *Internet Draft* citado. [6]

XSSP es un protocolo que permite a un agente XSDF suscribirse a los cambios en la información de servicio de otro agente XSDF. Esto permite a los agentes XSDF que requieren una versión actualizada de determinados servicios, recibir notificaciones asíncronas con las modificaciones de dichos servicios. Sin XSSP, estos agentes XSDF necesitarían solicitar la información de los servicios periódicamente. En particular, los DAs de un mismo ámbito, utilizan XSSP para mantener sus bases de datos de servicios sincronizadas.

XSSP sigue un sencillo modelo cliente-servidor, en el que el cliente solicita ser suscrito a los cambios que se realicen en la información de servicio del servidor. Cuando se produce algún cambio en esta información, se notifica al cliente.

Hay que tener en cuenta que cualquier agente XSDF puede ser un cliente XSSP, pero únicamente los agentes XSDF que poseen información de servicios (SAs y DAs) pueden ser servidores XSSP. Lógicamente los SAs sólo notificarán sobre los cambios producidos en sus servicios. En el caso de los DAs, existen dos tipos de suscripción:

- **Suscripción local**: A los suscriptores únicamente se les notifican los cambios en el DA producidos por la información recibida de un SA directamente mediante XSRP. Para evitar bucles, los cambios producidos por la información recibida de otro DA del mismo reino no se notifican a los suscriptores.
- **Suscripción global**: Los suscriptores son notificados mediante XSSP de todos los cambios realizados en la información de los servicios del DA, sin hacer distinción sobre el origen de los cambios.

En relación a la información del servicio a la que es posible suscribirse, existen tres tipos de suscripciones:

- **Por reino**: El agente XSDF recibirá notificaciones en caso de que se realicen cambios en cualquiera de los servicios de un determinado reino.
- **Por tipo de servicio**: El agente XSDF recibirá notificaciones en caso de que se realicen cambios en un determinado tipo de servicio.
- **Por ID de servicio**: El agente XSDF recibirá notificaciones en caso de que se realicen cambios en un determinado servicio con un ID específico.

El agente XSDF que recibirá las notificaciones también se modela como un servicio denominado servicio de notificación. Normalmente los agentes XSDF que recibirán notificaciones son los mismos que han solicitado la operación XSSP, pero se puede especificar otro. También se pueden solicitar notificaciones *multicast*.

En XSSP las suscripciones no indican únicamente cual es el conjunto de servicios objetivo, sino también qué tipo de cambios se deberían notificar. Esos tipos se definen en base a las diferentes

operaciones XSRP, ya que la mayor parte de los cambios se realizan debido a operaciones XSRP recibidas por los DAs.

4.1. FORMATO DE LOS MENSAJES

Este apartado describe el formato de todos los mensajes XSSP. Estos mensajes se intercambian entre un cliente y un servidor XSSP.

Todos los mensajes empleados por los protocolos que conforman XSDF están codificados mediante XBE32 [9] para un uso más eficiente de los recursos de red. A pesar de ello, para mejorar la comprensión de los mensajes, se representarán en el código XML equivalente.

4.1.1. ELEMENTO XSSPV1

Todos los mensajes de XSDF tienen una estructura común: un único elemento que contiene la cabecera, y uno o varios elementos correspondientes a las operaciones. En los mensajes XSSP, a este elemento raíz se le denomina XSSPv1.

```
<xsspv1>
    <header></header>
    <*XSSP operation*></*XSSP operation*>
    ...
    <*XSSP operation*></*XSSP operation*>          (opcional)
    <signatureInformation></signatureInformation> (opcional)
</xsspv1>

Elemento XSSPv1:
 Nombre del elemento : xsspv1
 Tipo XBE32          : 0x0b01
```

Ilustración 24. Elemento XSSPv1

Las posibles operaciones que puede contener un mensaje XSSPv1 son las siguientes:

- Elementos "*Subscribe Service"*, "*Update Subscription"* y "*Unsubscribe S*ervice": Son solicitudes enviadas por el cliente con el objetivo de crear, modificar o eliminar una suscripción a servicios.
- Elementos "*Subscribe Service ACK*", "*Update Subscription ACK*" y "*Unsubscribe Service ACK*": Son enviados por el servidor para responder a las solicitudes enviadas por el cliente que hayan sido procesadas correctamente.
- Elemento "*Error*": Los envía el servidor para responder a las solicitudes enviadas por el cliente que hayan sido procesadas erróneamente.
- Elemento "*Service E*vent": El servidor envía estos mensajes cuando se produce algún evento de creación, modificación o eliminación en alguno de los servicios a los que se ha suscrito el cliente.

4.1.2. ELEMENTO SUBSCRIBE SERVICE

Un cliente XSSP envía esta solicitud a un servidor XSSP con el fin de crear una suscripción. Los eventos de los servicios incluidos en el elemento "*Subscription Target*" deberían enviase al servicio de notificación indicado.

```
<subscribeService>
    <subscriptionId></subscriptionId>
    <subscriptionTarget></subscriptionTarget>
    <service></service>
    <subscribeInformation></subscribeInformation>
    <refreshInformation></refreshInformation>       (opcional)
</subscribeService>

Elemento Subscribe Service:
 Nombre del elemento : subscribeService
 Tipo XBE32          : 0x0b10
```

Ilustración 25. Elemento Subscribe Service

El atributo "*Subscription Identifier*" permite identificar la suscripción con el objetivo de que un mismo cliente XSSP pueda tener varias suscripciones en el mismo servidor XSSP. En la versión original del draft de XSSP este atributo no existía, sino que el identificador del servicio de notificación también se empleaba para identificar la suscripción, lo que hacía imposible que un mismo cliente XSSP tuviera múltiples suscripciones.

```
Atributo Subscription Identifier:
 Nombre del elemento : subscriptionId
 Tipo XBE32          : 0x3804
```

Ilustración 26. Atributo Subscription Identifier

El elemento "*Subscription Target*" debe contener a qué tipo de servicios quiere suscribirse el cliente. Si está vacío, el servicio de notificación debe suscribirse a todos los servicios de los ámbitos especificados en la cabecera.

Los clientes XSSP deben especificar ciertas propiedades al crear una suscripción. El elemento "*Subscribe Information*" debe incluir si la suscripción es global o local y qué tipo de eventos deberían ser notificados.

Cuando se va a crear una suscripción, el elemento "*Service*" debe contener la siguiente información sobre el servicio de notificación: ID, estado, información principal y localización. El resto de información es opcional y puede no incluirse.

Los clientes XSSP pueden incluir un elemento "*Refresh Information*" para sugerir a los servidores XSSP cuál es el intervalo de actualización apropiado para la suscripción. Sin embargo, un cliente XSSP debe usar el intervalo de actualización definido por el servidor XSSP en la respuesta, sea o no igual al sugerido.

4.1.3. ELEMENTO SUBSCRIBE SERVICE ACKNOLEDGMENT

Esta operación es enviada por un servidor XSSP para confirmar una operación "*Subscribe Service*" solicitada por un cliente XSSP.

```
<subscribeServiceAck>
    <subscriptionId></subscriptionId>
    <subscriptionTarget></subscriptionTarget>
    <service></service>
    <refreshInformation></refreshInformation>
</subscribeServiceAck>

Elemento Subscribe Service Acknowledgment:
 Nombre del elemento : subscribeServiceAck
 Tipo XBE32          : 0x0b11
```

Ilustración 27. Elemento Subscribe Service ACK

El atributo "*Subscription Identifier*" permite identificar la suscripción con el objetivo de que un mismo cliente XSSP pueda tener varias suscripciones en el mismo servidor XSSP.

El elemento "*Subscription Target*" debe contener a qué tipo de servicios se ha suscrito el servicio de notificación. Si está vacío, se ha suscrito a todos los servicios de los ámbitos definidos en la cabecera.

El elemento "*Service*" debe contener el ID del servicio de notificación que ha sido suscrito. El resto de información es opcional y no debería incluirse.

El servidor XSSP define en el elemento "*Refresh Information*" el intervalo de actualización que debería seguir el cliente XSSP. Si no se recibe un "*Update Subscription*" antes de que expire el intervalo, la suscripción se eliminará.

4.1.4. ELEMENTO UPDATE SUBSCRIPTION

Un cliente XSSP envía esta operación para actualizar una suscripción en un servidor XSSP. Además, puede actualizar la información del servicio de notificación y los detalles de la suscripción.

```
<updateSubscription>
    <subscriptionId></subscriptionId>
    <subscriptionTarget></subscriptionTarget>
    <subscribeInformation></subscribeInformation>    (opcional)
    <service></service>
    <refreshInformation></refreshInformation>        (opcional)
</updateSubscription>

Elemento Update Subscription:
 Nombre del elemento : updateSubscription
 Tipo XBE32          : 0x0b20
```

Ilustración 28. Elemento Update Subscription

El atributo "*Subscription Identifier*" permite identificar la suscripción con el objetivo de que un mismo cliente XSSP pueda tener varias suscripciones en el mismo servidor XSSP.

El elemento "*Subscription Target*" debe incluir a qué tipo de servicios está suscrito el Agente XSSP. Si está vacío, se debería actualizar la suscripción a todos los ámbitos definidos en la cabecera.

También puede incluirse un elemento "*Subscribe Information*" para actualizar las propiedades de la suscripción. Los atributos de una suscripción global no pueden ser distintos a los originales. Sin embargo, los eventos a notificar sí pueden ser cambiados.

El elemento "*Service*" debe incluir el ID y elemento "*State*" del servicio de notificación suscrito. Es posible incluir información adicional con el objetivo de cambiar la suscripción. En ese caso, los atributos adecuados del elemento "*MetaInfo*" deben incrementarse.

Los clientes XSSP pueden incluir un elemento "*Refresh Information*" para sugerir a los servidores XSSP cuál es el intervalo de actualización apropiado para la suscripción. Sin embargo, un cliente XSSP debe usar el intervalo de actualización definido por el servidor XSSP en la respuesta, sea o no igual al sugerido.

4.1.5. ELEMENTO UPDATE SUBSCRIPTION ACKNOLEDGMENT

Un servidor XSSP utiliza esta operación para confirmar una operación "*Update Subscription*" solicitada por un cliente XSSP.

```
<updateSubscriptionAck>
    <subscriptionId></subscriptionId>
    <subscriptionTarget></subscriptionTarget>
    <service></service>
    <refreshInformation></refreshInformation>
</updateSubscriptionAck>

Elemento Update Subscription Acknowledgment:
 Nombre del elemento : updateSubscriptionAck
 Tipo XBE32          : 0x0b21
```

Ilustración 29. Elemento Update Subscription ACK

El atributo "*Subscription Identifier*" permite identificar la suscripción con el objetivo de que un mismo cliente XSSP pueda tener varias suscripciones en el mismo servidor XSSP.

El elemento "*Subscription Target*" debe incluir a qué tipo de servicios está suscrito el servicio de notificación. En caso de estar vacío, el servicio estará suscrito a todos los ámbitos incluidos en la cabecera.

El elemento "*Service*" debe incluir el ID del servicio de notificación cuya suscripción ha sido actualizada. El resto de información es opcional y no debería aparecer.

El servidor XSSP define en el elemento "*Refresh Information*", el intervalo de actualización que debería seguir el cliente XSSP. Si no se recibe un "*Update Subscription*" antes de que expire el intervalo, se eliminará la suscripción.

4.1.6. ELEMENTO UNSUBSCRIBE SERVICE

Con este mensaje, un cliente XSSP solicita a un servidor XSSP la eliminación de la suscripción de un servicio de notificación específico.

```
<unsubscribeService>
    <subscriptionId></subscriptionId>
    <subscriptionTarget></subscriptionTarget>
    <service></service>
</unsubscribeService>

Elemento Unsubscribe Service:
 Nombre del elemento : unsubscribeService
 Tipo XBE32          : 0x0b30
```

Ilustración 30. Elemento Unsubscribe Service

El atributo "*Subscription Identifier*" permite identificar la suscripción con el objetivo de que un mismo cliente XSSP pueda tener varias suscripciones en el mismo servidor XSSP.

El elemento "*Subscription Target*" debe contener a qué tipo de servicios está suscrito el cliente XSSP. En caso de estar vacío, se debería eliminar la suscripción a todos los ámbitos incluidos en la cabecera.

El elemento "*Service*" debe incluir el ID del servicio suscrito. El resto de información es opcional y puede no aparecer.

4.1.7. ELEMENTO UNSUBSCRIBE SERVICE ACKNOLEDGMENT

Un servidor XSSP emplea esta operación para confirmar la operación "*Unsubscribe Service*" solicitada por el cliente XSSP.

```
<unsubscribeServiceAck>
    <subscriptionId></subscriptionId>
    <subscriptionTarget></subscriptionTarget>
    <service></service>
</unsubscribeServiceAck>

Elemento Unsubscribe Service Acknowledgment:
 Nombre del elemento : unsubscribeServiceAck
 Tipo XBE32          : 0x0b31
```

Ilustración 31. Elemento Unsubscribe Service ACK

El atributo "*Subscription Identifier*" permite identificar la suscripción con el objetivo de que un mismo cliente XSSP pueda tener varias suscripciones en el mismo servidor XSSP.

El elemento "*Subscription Target*" debe contener a qué tipo de servicios estaba registrado el cliente XSSP. En caso de estar vacío, se habrá eliminado la suscripción a todos los ámbitos incluidos en la cabecera.

El elemento "*Service*" debe contener el ID del servicio de notificación del que se ha eliminado la suscripción. El resto de información es opcional y no debería aparecer.

4.1.8. ELEMENTO ERROR

XSDF define un elemento "*Error*" genérico para los errores de todos los protocolos. Los siguientes errores son los que puede generar un servidor XSSP:

Código de error	Nombre del error	Causa
0x000b0001	SUBSCRIPTION_COLLISION	ID de la suscripción repetido.
0x000b0002	SUBSCRIPTION_NOT_FOUND	Suscripción no encontrada.
0x000b0003	UNSUPPORTED_PROTOCOL	Protocolo de notificación no soportado.
0x000b0004	INVALID_SUBSCRIPTION	Suscripción local o global no válida.

Tabla 2. Elementos Error de XSSP

Un error de XSSP se codifica en un único elemento "Error" que incluye el código de error, un nombre y otros elementos opcionales. Todos los elementos "Error" de XSSP deben incluir un atributo "*subscriptionId*" con el ID de la suscripción ha causado el error.

Un elemento "*Error*" de tipo "*UNSUPPORTED_PROTOCOL*" debería además incluir un elemento "*Service*" con los protocolos de notificación soportados por el servidor mediante uno o varios elementos "*Protocol*".

Un elemento "*Error*" de tipo "*INVALID_SUBSCRIPTION*" debería además incluir los elementos "*Subscribe Information*" que están en conflicto con el solicitado.

4.1.9. ELEMENTO SERVICE EVENT

El servidor XSSP utiliza esta operación para notificar a un suscriptor de que un servicio al que estaba suscrito ha sido creado, modificado o eliminado.

A pesar de no aparecer en el Draft original, que empleaba mensajes XSLP o XSTP, ha sido necesario el diseño y uso de este elemento para notificar a los clientes de manera sencilla e incluir toda la información necesaria de las suscripciones, como por ejemplo el ID de la suscripción.

```
<serviceEvent>
    <eventType></eventType>
    <record></record>
</serviceEvent>

Elemento Service Event:
 Nombre del elemento : serviceEvent
 Tipo XBE32          : 0x0b40
```

Ilustración 32. Elemento Service Event

El atributo "*Event Type*" debe indicar qué tipo de cambio ha sufrido el servicio.

```
Atributo Event Type:
 Nombre del atributo : eventType
 Tipo XBE32          : 0x2d05
```

Ilustración 33. Atributo Event Type

Los valores que el atributo "*Event Type*" puede tomar son los siguientes:

Evento	Valor
Servicio registrado	0x01
Servicio actualizado	0x02
Información del servicio actualizada	0x03
Servicio eliminado	0x04
Servicio expirado	0x05

Tabla 3. Tipos de eventos XSSP en elemento Service Event

El elemento "*Record*" debe contener la información de servicio necesaria por cada evento. En el caso de un evento de registro, se debe incluir el servicio completo, así como la información relacionada con el tiempo de vida del servicio.

En el caso de un evento de actualización, se deben incluir el ID y las partes del servicio que hayan sido modificadas, así como la información relacionada con el tiempo de vida del servicio.

En el caso de un evento de eliminación, solo se debe incluir el ID del servicio.

4.1.10. PARÁMETROS DE OPERACIONES COMUNES EN XSSP

Cuando se crea o actualiza una suscripción, el agente XSDF debe especificar el servicio de notificación y cierta información adicional. Esta sección define esa información incluida en las solicitudes "*Subscribe Service*" y "*Update Subscription*". Además, describe el elemento "*Subscription Target* " que aparece en todas las operaciones XSSP.

4.1.10.1. ELEMENTO SUBSCRIPTION TARGET

El cliente XSSP indica con este elemento en qué servicios está interesado. Puede incluir el ámbito, el tipo o el ID exacto de los servicios a los que se quiere suscribir. En caso de especificar un ID, también se debe incluir el tipo de servicio.

```
<subscriptionTarget>
    <realm></realm>             (opcional)
    <serviceType></serviceType> (opcional)
    <serviceId></serviceId>     (opcional)
</subscriptionTarget>

Elemento Subscription Target:
 Nombre del elemento : subscriptionTarget
 Tipo XBE32          : 0x0821
```

Ilustración 34. Elemento Subscription Target

El elemento "*Realm*" puede contener el conjunto de ámbitos del reino a los que se quiere suscribir. En caso de que el elemento esté vacío y no se haya especificado un tipo de servicio, la suscripción debería realizarse sobre todos los servicios de los ámbitos definidos en la cabecera del mensaje.

Los clientes XSSP pueden suscribirse a todos los servicios de un tipo específico del reino. Si el elemento "*Type*" incluye algún elemento "*Path*", el servidor XSSP debería notificar cambios en los servicios que proporcionen la funcionalidad solicitada.

Cuando aparece, el atributo "*Service Ids*" indica a qué servicios concretos está suscrito el cliente XSSP. En caso de que esta lista incluya servicios desconocidos, los IDs deberían recordarse por si se registran en el futuro y la solicitud debería aceptarse de todas formas.

4.1.10.2. ELEMENTO SUBSCRIBE INFORMATION

Este elemento describe las propiedades del servicio de suscripción. El elemento "*Subscribe Information*" debe incluirse en una solicitud "*Subscribe Service*". La información de evento puede modificarse más adelante mediante una solicitud "*Update Subscription*".

```
<subscribeInfo>
    <globalSubscription></globalSubscription>
    <eventInfo></eventInfo>
</subscribeInfo>

Elemento Subscribe Information:
 Nombre del elemento : subscribeInfo
 Tipo XBE32          : 0x0420
```

Ilustración 35. Elemento Subscribe Information

El atributo "*Global Subscription*" indica si el DA debería notificar únicamente los cambios locales realizados a los servicios o si también debería notificar los cambios recibidos por otros DAs. Los SAs únicamente soportan suscripciones locales y deberían rechazar suscripciones globales, respondiendo con un error de tipo "*INVALID_SUBSCRIPTION*".

```
Atributo Global Subscription:
 Nombre del atributo : globalSubscription
 Tipo XBE32          : 0x2655
```

Ilustración 36. Atributo Global Subscription

El atributo "*Event Information*" contiene un conjunto de valores para indicar a qué tipos de eventos quiere estar suscrito.

```
Atributo Event Information:
 Nombre del atributo : eventInfo
 Tipo XBE32          : 0x2C02
```

Ilustración 37. Atributo Event Information

Los posibles valores para el atributo "*Event Information*" son:

Evento	Valor
Servicio registrado	0x01
Servicio actualizado	0x02
Información del servicio actualizada	0x03
Servicio eliminado	0x04
Servicio expirado	0x05

Tabla 4. Tipos de eventos XSSP en elemento Subscribe Information

4.1.10.3. ELEMENTO REFRESH INFORMATION

Este elemento permite a un servidor XSDF controlar el tiempo de actualización del registro de un servicio, o de una actualización.

```
<refreshInfo>
    <minimumLife></minimumLife>
    <maximumLife></maximumLife>
</refreshInfo>

Refresh Information Element:
 Element Name: refreshInfo
 XBE32 Type: 0x0521
```

Ilustración 38. Elemento Refresh Information

El atributo "*Minimum Life*" indica el tiempo en milisegundos durante el cual la información asociada no debería ser actualizada.

```
Atributo Minimum Life:
 Nombre del atributo   : minLife
 Tipo XBE32            : 0x3253
```

Ilustración 39. Atributo Minimum Life

El atributo "*Maximum Life*" indica el tiempo en milisegundos durante el cual la información asociada se mantendrá registrada. Si no se recibe un mensaje de actualización durante este tiempo, el registro de servicio o la suscripción será eliminado.

```
Atributo Maximum Life:
 Nombre del atributo   : maxLife
 Tipo XBE32            : 0x3254
```

Ilustración 40. Atributo Maximum Life

Los clientes XSSP deberían actualizar sus suscripciones en un tiempo aleatorio entre los valores de los dos atributos.

4.1.11. PARÁMETROS DE OPERACIONES COMUNES EN XSDF

Los elementos y atributos que se describirán en esta sección, son comunes para todos los protocolos de XSDF. La fuente que se ha empleado para su descripción es [8].

4.1.11.1. ELEMENTO HEADER

Este elemento define la cabecera común a todos los mensajes XSDF. Identifica tanto a los extremos como a la transacción. Cuando se responde un mensaje se intercambian los elementos de origen y destino.

```
<header>
    <xid></xid>
    <realm></realm>
    <source></source>
    <destination></destination>
    <authInfo></authInfo>            (opcional)
</header>

Elemento Header:
 Nombre del elemento : header
 Tipo XBE32          : 0x0810
```

Ilustración 41. Elemento Header

El atributo "*xid*" contiene un identificador único para la transacción XSDF que se genera de forma aleatoria por el cliente. Al responder un mensaje se mantiene el mismo valor para que se puedan relacionar petición y respuesta rápidamente.

```
Atributo Transaction Identifier:
 Nombre del atributo   : xid
 Tipo XBE32            : 0x3281
```

Ilustración 42. Atributo Xid

4.1.11.2. ELEMENTO REALM

Este elemento identifica el dominio y los ámbitos en los que se debería aplicar las operaciones XSDF contenidas en el mensaje.

```
<realm>
    <domain></domain>   (opcional)
    <scope></scope>
    ...
    <scope></scope>     (opcional)
</realm>

Elemento Realm:
 Nombre del elemento : realm
 Tipo XBE32          : 0x0700
```

Ilustración 43. Elemento Realm

El atributo "*Domain*" contiene un identificador jerárquico de la organización que provee los servicios. Permite emplear la tecnología DNS para encontrar a los DAs.

```
Atributo Domain:
 Nombre del atributo   : domain
 Tipo XBE32            : 0x2871
```

Ilustración 44. Atributo Domain

El atributo "*Scope*" contiene el nombre de un ámbito que pertenece al dominio.

```
Atributo Scope:
 Nombre del atributo   : scope
 Tipo XBE32            : 0x2872
```

Ilustración 45. Atributo Scope

4.1.11.3. ELEMENTO ENDPOINT

Este tipo de elemento identifica uno de los extremos del mensaje XSDF. El elemento "*Source*" identifica al agente XSDF y/o al usuario que ha generado el mensaje. No debe ser un valor reservado ("Todos los agentes XSDF" o "Agente XSDF Desconocido"). El elemento "*Destination*" identifica al agente XSDF y/o al usuario al que va dirigido el mensaje. Puede ser un valor reservado ("Todos los agentes XSDF" o "Agente XSDF Desconocido").

```
<endpoint>
    <service></service>      (opcional)
    <user></user>            (opcional)
</endpoint>

Elemento Source Endpoint:
 Nombre del elemento : source
 Tipo XBE32          : 0x0811

Elemento Destination Endpoint:
 Nombre del elemento : destination
 Tipo XBE32          : 0x0812
```

Ilustración 46. Elemento Endpoint

El elemento "*Service*" identifica el agente XSDF de origen o destino del mensaje. Sin embargo, puede no aparecer, ya que los UAs no tienen por qué tener un servicio asociado. En caso de existir, el elemento "*Service*" debe incluir el identificador del servicio, y puede incluir el estado del servicio. El resto de información no debería ser incluida.

El elemento "*User*" contiene información sobre el usuario responsable del agente XSDF de origen o destino.

4.1.11.4. ELEMENTO SERVICE

El elemento "*Service*" es el más importante de los elementos de XSDF, ya que contiene toda la información sobre un servicio. Esto incluye: ID, tipo, localización, protocolos que pueden utilizarse, su estado, etc.

```
<service>
    <id></id>
    <serviceState></serviceState>                    (opcional)
    <serviceMainInfo></serviceMainInfo>              (opcional)
    <serviceLocationInfo></serviceLocationInfo>      (opcional)
    <serviceAddInfo></serviceAddInfo>                (opcional)
</service>

Elemento Service:
 Nombre del elemento : service
 Tipo XBE32          : 0x0100
```

Ilustración 47. Elemento Service

El atributo "*Service Identifier*" es obligatorio e incluye un valor de 16 octetos que sirven como un identificador único para el servicio.

```
Atributo Service Identifier:
 Nombre del atributo   : id
 Tipo XBE32            : 0x3511
```

Ilustración 48. Atributo Service Identifier

Los siguientes valores para el atributo están reservados y no deberían utilizarse para identificar ningún servicio:

ID	Significado
00000000-0000-0000-0000-000000000000	Agente XSDF desconocido
ffffffff-ffff-ffff-ffff-ffffffffffff	Todos los agentes XSDF

Tabla 5. Identificadores de servicio reservados

4.1.11.5. ELEMENTO SERVICE STATE

Este elemento contiene toda la información temporal de un servicio.

```
<serviceState>
    <metaInfo></metaInfo>
    <selectState></selectState>    (opcional)
</serviceState>

Elemento Service State:
 Nombre del elemento : serviceState
 Tipo XBE32          : 0x0110
```

Ilustración 49. Elemento Service State

4.1.11.6. ELEMENTO META-INFORMATION

Este elemento contiene la versión de la información relativa a todos los elementos informativos del servicio. Comparando dos elementos "*metaInfo*" de un mismo servicio, un agente puede diferenciar cual es más reciente y qué elementos han sido modificados.

```
<metaInfo>
    <stateTimestamp></stateTimestamp>
    <mainInfoSeqNum></mainInfoSeqNum>           (opcional)
    <locationInfoSeqNum></locationInfoSeqNum>   (opcional)
    <addInfoSeqNum></addInfoSeqNum>             (opcional)
</metaInfo>

Elemento Meta-Information:
 Nombre del elemento : metaInfo
 Tipo XBE32          : 0x0111
```

Ilustración 50. Elemento Meta-Information

El atributo "*State Timestamp*" indica el momento en el que el elemento ha sido generado. Este tiempo puede obtenerse de un reloj real o de un reloj lógico. El único requisito es que un estado más reciente posea un valor de tiempo mayor.

```
Atributo State Timestamp:
 Nombre del atributo   : stateTimestamp
 Tipo XBE32            : 0x331a
```

Ilustración 51. Atributo State Timestamp

Cuando se modifica un elemento del servicio, su contador relacionado debe ser incrementado en una unidad, siendo 0 el valor inicial. Dichos contadores son encapsulados en los atributos "*Main Info Sequence Number*", "*Location Info Sequence Number*" y "*Additional Info Sequence Numbe*r".

```
Atributo Main Info Sequence Number:
 Nombre del atributo   : mainInfoSeqNum
 Tipo XBE32            : 0x321b
```

Ilustración 52. Atributo Main Info Sequence Number

```
Atributo Location Info Sequence Number:
 Nombre del atributo   : locationInfoSeqNum
 Tipo XBE32            : 0x321c
```

Ilustración 53. Atributo Location Info Sequence Number

```
Atributo Additional Info Sequence Number:
 Nombre del atributo   : addInfoSeqNum
 Tipo XBE32            : 0x321d
```

Ilustración 54. Atributo Additional Info Sequence Number

4.1.11.7. ELEMENTO SELECTION STATE

Este elemento contiene toda la información temporal que necesita un algoritmo de selección para medir la calidad del servicio. Para poder escoger un servicio u otro según su calidad con precisión, es necesario obtener una medida reciente de sus estados.

```
<selectState>
    <resources></resources>     (opcional)
    <workload></workload>       (opcional)
</selectState>

Elemento Selection State:
 Nombre del elemento : selectState
 Tipo XBE32          : 0x0311
```

Ilustración 55. Elemento Select State

El atributo "*Resources*" indica la cantidad de recursos disponibles para los clientes que quieran usar el servicio. Existe una política de selección que elige un servicio u otro en función de este valor. Dado que la cantidad de recursos disponibles no puede ser menor que cero, este atributo no podrá tener un valor negativo. Además, si el valor es cero, indica que el servicio no está disponible temporalmente y no debe ser utilizado por los usuarios sea cual sea su política de selección.

```
Atributo Resources:
 Nombre del atributo   : resources
 Tipo XBE32            : 0x3231
```

Ilustración 56. Atributo Resources

El atributo "*Workload*" indica la cantidad de recursos ocupados por los usuarios que están accediendo al servicio en un momento determinado. Cuanto más bajo sea el valor, mayor es la capacidad de atender a nuevos usuarios. Dado que la cantidad de recursos consumidos no puede ser menor que cero, este atributo no podrá tener un valor negativo. Existe una política de selección que elige un servicio u otro en función de este valor. Cada servicio puede definir su propia medida de carga. Por defecto, la carga de trabajo de un servicio es igual a la carga porcentual de la CPU del servidor que lo aloja.

```
Atributo Workload:
 Nombre del atributo   : workload
 Tipo XBE32            : 0x3232
```

Ilustración 57. Atributo Workload

4.1.11.8. ELEMENTO SERVICE MAIN INFORMATION

Este elemento especifica el tipo del servicio. Puede incluir el nombre del servicio en texto legible por humanos, junto con la política de selección que debería ser empleada para escoger entre servicios de este tipo.

```
<serviceMainInfo>
    <serviceType></serviceType>
    <alias></alias>                 (opcional)
    <selectInfo></selectInfo>       (opcional)
</serviceMainInfo>

Elemento Service Main Information:
 Nombre del elemento : serviceMainInfo
 Tipo XBE32          : 0x0120
```

Ilustración 58. Elemento Service Main Information

El atributo "*Alias*" es opcional y permite identificar un servicio por nombre en lugar de por un ID para ser más comprensible por usuarios.

```
Atributo Alias:
 Nombre del atributo   : alias
 Tipo XBE32            : 0x2814
```

Ilustración 59. Atributo Alias

4.1.11.9. ELEMENTO SERVICE TYPE

Este elemento identifica el tipo del servicio. Si un servicio se encuentra distribuido entre varios servidores, este elemento identificaría también qué partes del servicio se encuentran en esta instancia.

```
<serviceType>
    <type></type>
    <path></path>   (opcional)
    ...
    <path></path>   (opcional)
</serviceType>

Elemento Service Type:
 Nombre del elemento : serviceType
 Tipo XBE32          : 0x0121
```

Ilustración 60. Elemento Service Type

El atributo "*Type*" identifica qué tipo de servicio se proporciona, como por ejemplo "printer" en el caso de una impresora, o "xsdf:da" en el caso de un DA.

```
Atributo Type:
 Nombre del atributo   : type
 Tipo XBE32            : 0x2812
```

Ilustración 61. Atributo Type

Si este tipo de servicio puede proporcionar recursos con estructura jerárquica (como un servidor de ficheros), los atributos "*Path*" especifican que ramificación del árbol de recursos se proporcionan por esta instancia del servicio. El valor por defecto es "/", la raíz del árbol.

```
Atributo Path:
 Nombre del atributo   : path
 Tipo XBE32            : 0x2813
```

Ilustración 62. Atributo Path

4.1.11.10. ELEMENTO SELECTION INFORMATION

Este elemento contiene toda la información estática que necesita un algoritmo de selección para evaluar la calidad del servicio.

```
<selectInfo>
    <policies></policies>
    <priority></priority>   (opcional)
    <weight></weight>       (opcional)
</selectInfo>

Elemento Selection Information:
 Nombre del elemento : selectInfo
 Tipo XBE32          : 0x0321
```

Ilustración 63. Elemento Select Information

El atributo "*Policies*" contiene una lista de políticas de selección que pueden ser utilizadas para escoger entre servicios del mismo tipo.

```
Atributo Policies:
 Nombre del atributo   : policies
 Tipo XBE32            : 0x3133
```

Ilustración 64. Atributo Policies

Los valores disponibles para el atributo para los distintos tipos de algoritmos son:

Valor	Política de Selección
0x0000	*None*
0x0001	*Round Robin*
0x0002	*Least Used*
0x0003	*Most Resources*
0x0004	*Closest*

Tabla 6. Políticas de selección

El atributo "*Priority*" es opcional y contiene la prioridad de la instancia del servicio. Los algoritmos de selección deberían aplicarse únicamente sobre los servicios con la prioridad más alta. Este mecanismo facilita el despliegue de servicios con alta disponibilidad. El valor por defecto para este campo es 0, pudiendo admitir valores negativos.

```
Atributo Priority:
 Nombre del atributo   : priority
 Tipo XBE32            : 0x3234
```

Ilustración 65. Atributo Priority

El atributo "*Weight*" es opcional y especifica la eficiencia relativa del servicio comparado con otros servicios del mismo tipo. Este valor lo emplea la política de selección Round Robin para mejorar el reparto de carga entre servicios iguales, pero con distinta capacidad de procesamiento. El valor siempre debe ser positivo.

```
Atributo Weight:
 Nombre del atributo   : weight
 Tipo XBE32            : 0x3235
```

Ilustración 66. Atributo Weight

4.1.11.11. ELEMENTO SERVICE LOCATION INFORMATION

Este elemento contiene toda la información necesaria para contactar con el proceso de un determinado servicio de un servidor.

```
<serviceLocationInfo>
    <inet></inet>
    <protocol></protocol>
    ...
    <protocol></protocol>   (opcional)
</serviceLocationInfo>

Elemento Service Location Information:
 Nombre del elemento : serviceLocationInfo
 Tipo XBE32          : 0x0130
```

Ilustración 67. Elemento Service Location Information

4.1.11.12. ELEMENTO INET

Este elemento contiene la localización del servidor en la red, así como funciones adicionales de la capa de red.

```
<inet>
    <ipv4 addresses></ipv4 addresses>    (opcional)
    <ipv6 addresses></ipv6 addresses>    (opcional)
    <hostname></hostname>                (opcional)
    ...
    <hostname></hostname>                (opcional)
    <capabilities16></capabilities16>    (opcional)
</inet>

Elemento Inet:
 Nombre del elemento : inet
 Tipo XBE32          : 0x0131
```

Ilustración 68. Elemento Inet

El atributo "*Capabilities16*" es opcional y contiene la lista de funciones adicionales de la pila de red del servidor que proporciona el servicio.

```
Atributo Capabilities16:
 Nombre del atributo   : capabilities16
 Tipo XBE32            : 0x3118
```

Ilustración 69. Atributo Capabilities16

4.1.11.13. ELEMENTO PROTOCOL

Este elemento identifica un protocolo que puede ser empleado para acceder al servicio. Incluye puertos de comunicación y funciones adicionales del protocolo.

```
<protocol>
    <name></name>
    <transPorts></transPorts>
    <capabilities16></capabilities16>    (opcional)
</protocol>

Elemento Protocol:
 Nombre del elemento : protocol
 Tipo XBE32          : 0x0132
```

Ilustración 70. Elemento Protocol

Los protocolos son identificados por su nombre, por lo que no deberían existir dos protocolos diferentes con el mismo nombre.

El atributo "*Transport Protocols & Ports*" contiene una lista de los puertos de comunicación y el protocolo de transporte que el servidor acepta. Los dos últimos octetos representan el puerto, mientras que los dos primeros representan el protocolo.

```
Atributo Transport Protocols & Ports:
 Nombre del atributo   : transPorts
 Tipo XBE32            : 0x321a
```

Ilustración 71. Atributo Transport Protocols & Ports

4.1.11.14. ELEMENTO SERVICE ADITIONAL INFORMATION

Este elemento contiene información adicional sobre el servicio.

```
<serviceAddInfo>
    <vendor></vendor>                (opcional)
    <vendorURL></vendorURL>          (opcional)
    <model></model>                  (opcional)
    <modelURL></modelURL>            (opcional)
    <version></version>              (opcional)
    <url></url>                      (opcional)
    <description></description>      (opcional)
    ...
    <description></description>      (opcional)
    <icon></icon>                    (opcional)
    ...
    <icon></icon>                    (opcional)
    <operator></operator>            (opcional)
    ...
    <operator></operator>            (opcional)
</serviceAddInfo>

Service Additional Information Element:
 Element Name : serviceAddInfo
 XBE32 Type   : 0x0140
```

Ilustración 72. Elemento Service Additional Information

4.1.11.15. ELEMENTO ICON

Este elemento contiene información gráfica sobre un servicio. El único dato obligatorio es la URL que apunta a la imagen.

```
<icon>
    <url></url>
    <mimeType></mimeType>    (opcional)
    <width></width>          (opcional)
    <height></height>        (opcional)
    <depth></depth>          (opcional)
</icon>

Elemento Icon:
 Nombre del elemento : icon
 Tipo XBE32          : 0x0620
```

Ilustración 73. Elemento Icon

4.2. OPERACIONES XSSP

Todos los agentes XSDF emplean un procedimiento común para la generación, envío y procesamiento de mensajes. Esta sección describe los procedimientos concretos para los agentes XSSP.

XSSP es un protocolo cliente-servidor. Cualquier agente XSDF puede ser un cliente y enviar solicitudes de operación, mientras que solo los DAs y los SAs pueden ser servidores XSSP y procesar operaciones XSSP.

Todos los agentes XSSP deben soportar UDP como protocolo de transporte, aunque también pueden soportar opcionalmente TCP y SCTP. El puerto por defecto para todos los protocolos es el 725.

4.2.1. SUSCRIPCIÓN A UN SERVICIO

Los clientes XSSP se suscriben a servicios gestionados por servidores XSSP mediante el envío de una solicitud "*Subscribe Service*". Se pueden realizar varias suscripciones a la vez incluyendo varias solicitudes "*Subscribe Service* " en un mismo mensaje XSSP.

Una vez que el mensaje XSSP que contiene las operaciones "*Subscribe Service*" ha sido generado, se envía al servidor XSSP correspondiente. En el caso de las suscripciones globales de un ámbito, puede elegirse cualquier DA de ese ámbito. Si no es posible establecer una comunicación con ese DA, se puede probar con otro del mismo ámbito.

Cuando un servidor XSSP recibe un mensaje XSSP válido con operaciones "*Subscribe Service*" autorizadas, debe seguir los siguientes pasos para cada operación:

1. Si el elemento "*Subscription Target*" incluye algún ámbito, comprobar que aparezca también en la cabecera. En caso contrario, el proceso debe detenerse sin informar al cliente XSSP.
2. Comprobar que no existe una suscripción con el mismo ID. En caso de que ya exista una suscripción con ese ID, el proceso debe detenerse y se debe crear un mensaje de error "*SUBSCRIPTION_COLLISION*".
3. Comprobar si se soporta el protocolo de notificación incluido en el elemento "*Service Information*". En caso contrario, el proceso se debe detener y se debe generar un mensaje de error "*UNSUPPORTED_PROTOCOL*" que además indique los protocolos de notificación soportados.
4. Los SAs no deberían aceptar suscripciones globales ya que solo gestionan un pequeño conjunto de servicios. Si un SA recibe una solicitud "*Subscribe Service*" con el atributo "*Global Subscription*" activado, el proceso se debería detener y se debería generar un mensaje de error "*INVALID_SUBSCRIPTION*".
5. Si no se produce ningún error, el servidor XSSP debe registrar la suscripción en todos los ámbitos especificados y almacenar los datos sobre el servicio de notificación.
6. El servidor XSSP elige, teniendo en cuenta la sugerencia del cliente XSSP en caso de haberla, un intervalo de actualización y genera un "*Subscribe Service Ack*".
7. El servidor XSSP define la variable "*LAST_UPDATE_TIMESTAMP*" con el momento en que se produce la suscripción.
8. El servidor XSSP define un temporizador "*MAX_LIFE_TIMER*" asociado a la nueva suscripción. Dicho temporizado debe expirar pasado el tiempo decidido si no se ha recibido una actualización.

Para cada solicitud "*Subscribe Service*" que ha generado una nueva suscripción, el servidor XSSP debe generar una respuesta "*Subscribe Service Ack*".

Al recibir un mensaje XSSP que contiene uno o varios "*Subscribe Service Ack*", el cliente XSSP debe comprobar los atributos "*Minimum Life*" y "*Maximum Life*" para definir un temporizador "*REFRESH_TIMER*" asociado a la suscripción. El tiempo del temporizador en milisegundos debería elegirse aleatoriamente entre los valores de "*Minimum Life*" y "*Maximum Life*".

4.2.2. ACTUALIZACIÓN DE UNA SUSCRIPCIÓN

Cuando el temporizador "*REFRESH_TIMER*" asociado a una suscripción expira, el cliente XSSP debería actualizar la suscripción en todos los ámbitos a los que está suscrito. Para ello debe enviar una solicitud "*Update Subscription*" al servidor XSSP que gestiona la suscripción.

La información del servicio de notificación puede ser actualizada mediante el mismo mecanismo en cualquier momento posterior a que transcurra "*Minimum Life*" desde la última actualización.

Una vez se genera el mensaje XSSP que contiene la operación "*Update Subscription*", éste debe enviarse al servidor XSSP en el que se registró la suscripción. En caso de que se trate de una suscripción global de un ámbito y un DA no responda, se puede crear de nuevo la suscripción en otro DA del mismo reino.

Cuando un servidor XSSP recibe un mensaje XSSP válido con una o varias operaciones "*Update Subscription*", para cada operación se deben seguir los siguientes pasos:

1. Si el elemento "*Subscription Target*" incluye algún ámbito, comprobar que aparezca también en la cabecera. En caso contrario, el proceso debe detenerse sin informar al cliente XSSP.
2. Buscar la suscripción que se solicita actualizar mediante su ID de la suscripción. Si no se encuentra ninguna suscripción con ese ID, el proceso debe detenerse y debe generarse un mensaje de error "*SUBSCRIPTION_NOT_FOUND*".
3. Si una solicitud de actualización incluye un elemento "*Subscription Information*", hay que comprobar que el servicio de notificación haya sido registrado con el mismo atributo "*Global Subscription*". En caso contrario, el proceso debería detenerse y se debería generar un mensaje de error "*INVALID_SUBSCRIPTION*".
4. Si no se ha producido ningún error, el servidor XSSP debe actualizar la suscripción a todos los ámbitos registrados. Si se ha especificado información sobre el servicio de notificación, se debe actualizar la suscripción (con la excepción del protocolo de notificación).
5. El servidor XSSP elige, teniendo en cuenta la sugerencia del cliente XSSP en caso de haberla, un intervalo de actualización y genera un "*Update Subscription Ack*".
6. El servidor XSSP actualiza la variable "*LAST_UPDATE_TIMESTAMP*" con el momento en que se actualiza la suscripción.
7. El servidor XSSP pone a cero el temporizador "*MAX_LIFE_TIMER*" asociado a la suscripción actualizada. Dicho temporizado debe expirar pasado el tiempo decidido si no se ha recibido una actualización.

Para cada solicitud "*Update Subscription*" con la que la suscripción ha sido modificada, el servidor XSSP debe generar un "*Update Subscription Ack*".

Al recibir un mensaje XSSP que incluye uno o varios "*Update Subscription Ack*", el cliente XSSP debe comprobar los atributos "*Minimum Life*" y "*Maximum Life*" para definir un temporizador "*REFRESH_TIMER*" asociado a la suscripción. El tiempo del temporizador en milisegundos debería elegirse aleatoriamente entre los valores de "*Minimum Life*" y "*Maximum Life*".

4.2.3. ELIMINACIÓN DE UNA SUSCRIPCIÓN

Cuando un cliente XSSP quiere dejar de recibir notificaciones de un determinado servicio, puede eliminar su suscripción mediante el envío de una solicitud "*Unsubscribe Service*". Pueden

eliminarse múltiples registros en un solo mensaje XSSP incluyendo varias solicitudes "*Unsubscribe Service*" en el mismo.

Una vez que se generan las operaciones "*Unsubscribe Service*", se procede a enviarlas al servidor XSSP en el que se realizó la suscripción.

Cuando un servidor XSSP recibe un mensaje XSSP con una o varias operaciones "*Unsubscribe Service*", debe seguir los siguientes pasos con cada una de ellas:

1. Si el elemento "*Subscription Target*" incluye algún ámbito, comprobar que aparezca también en la cabecera. En caso contrario, el proceso debe detenerse sin informar al cliente XSSP.
2. Buscar la suscripción que se solicita eliminar mediante el ID de la suscripción. Si no se encuentra ninguna suscripción con dicho ID, el proceso se debe detener y se debe generar un mensaje de error "*SUBSCRIPTION_NOT_FOUND*".
3. Si no se ha producido ningún error, el servidor XSSP debe eliminar la suscripción en todos los ámbitos registrados. Esto se realiza borrando el temporizador "*MAX_LIFE_TIMER*" asociado, y eliminando la información del servicio de notificación y las propiedades de la suscripción.
4. Para cada solicitud "*Unsubscribe Service*" con la que se elimina una suscripción, el servidor XSSP debe generar una respuesta "*Unsubscribe Service Ack*". Toda respuesta que se envíe debe incluir el ID del servicio de notificación cuya suscripción se ha eliminado.

Cuando el temporizador "*MAX_LIFE_TIMER*" asociado a un registro expira, el servidor XSSP debe eliminar la suscripción de todos los ámbitos en los que estaba registrado.

4.2.4. NOTIFICACIÓN DE EVENTO

En la versión del protocolo utilizada para el desarrollo de este trabajo, XSSP solo se encarga de la gestión de suscripciones. Se debían utilizar otros protocolos para enviar las notificaciones. Dos de los protocolos XSDF que podían ser utilizados para enviar los eventos de servicio eran:

- XSLP: Cuando se emplea el protocolo XSLP, los servidores XSSP utilizan operaciones "*Service Advertisement*" XSLP para notificar los cambios en el servicio.
- XSTP: Cuando se emplea el protocolo XSTP, los servidores XSSP utilizan operaciones XSRP encapsuladas en mensajes XSTP para notificar los cambios en el servicio.

Ambos protocolos permiten la notificación *unicast* y *multicast* si se especifica la dirección apropiada en el elemento "*Service Location Information*". Siguiendo las reglas definidas, las notificaciones *multicast* deben incluirse en un mensaje XSDF que contenga 'todos los agentes XSDF' en el "*Destination Service Id*". Los mensajes *unicast* se envían al servicio de notificación con el ID adecuado.

Sin embargo, esta manera de transmitir modificaciones ha sido modificada para que pueda utilizarse el protocolo XSSP en lugar de XSLP. De este modo se puede transmitir información concreta relacionada con el evento que no podría ser transmitida con XSLP. Por ejemplo, para incluir el identificador de la suscripción, o para diferenciar si un evento de eliminación de un servicio se ha producido por la recepción de un mensaje, o porque su tiempo de vida ha expirado.

4.2.4.1. ANUNCIO DE SERVICIO XSSP

Para las suscripciones que emplean XSSP como protocolo de notificación, los servidores XSSP deberían enviar eventos de modificación con anuncios de servicio. Las operaciones "*Service Event*" pueden incluir la información completa de un servicio o solo la información que ha cambiado. En ambos casos, el elemento "*Service*" debe contener al menos su ID y el elemento "*Service State*". Es recomendable que los eventos de actualización contengan únicamente los subelementos del servicio actualizado y no toda la información del servicio.

Cuando se inicia un nuevo servicio, se debería incluir en el "*Service Event*" toda la información sobre el servicio, incluyendo el elemento "*Additional Information*". Si se detiene se debería generar un "*Service Event*" con TTL a 0.

4.3. CONSIDERACIONES DE SEGURIDAD

XSDF define mecanismos de autenticación que deberían utilizarse en caso de que no se use un procedimiento similar en una capa inferior de la pila de red.

Como medida de seguridad es recomendable el uso de todos los protocolos de XSDF en conjunto con TLS o IPsec. Si XSSP se utiliza en conjunto con TLS, el puerto por defecto para la comunicación mediante TCP o SCTP será el 726.

4.4. RESUMEN DE CAMBIOS

Durante el diseño y desarrollo de la implementación del protocolo XSSP se han realizado modificaciones que mejoran la versión original. Todos los cambios han sido aprobados por Manuel Urueña, tutor de este Trabajo Fin de Grado y autor de los drafts de XSDF, y serán incluidos en la próxima versión del protocolo.

1. Inclusión de un identificador de la suscripción en todos los mensajes XSSP, que permite a un cliente XSSP mantener varias suscripciones simultaneas en un mismo servidor XSSP. El cliente genera aleatoriamente este identificador antes de crear el mensaje de suscripción, y es validado por el servidor tras comprobar que no existe otra suscripción con ese identificador. Anteriormente, las suscripciones se identificaban mediante el ID del servicio de notificación del cliente XSSP, por lo que si un cliente XSSP quería realizar varias suscripciones tenía que crear un servicio de notificación diferente para cada una. Este procedimiento ha sido descartado por ser poco escalable.
2. En el elemento "*Subscribe Information*" se ha reemplazado la estructura original para que los eventos contenidos en el ahora atributo "*Event Information*" sean enviados más eficientemente por la red, sustituyendo la estructura original que contenía un elemento vacío por cada tipo de evento, por un vector de valores que identifican el tipo de evento.
3. Notificación de cambios en los servicios a los que está suscrito un cliente, mediante el propio protocolo XSSP. Anteriormente se contemplaba la posibilidad de que los eventos se notificaran mediante XSLP o XSTP. Notificar los eventos mediante XSLP no permitía el transporte de toda la información necesaria, por lo que se ha creado la operación XSSP "*Service Event*" y sus atributos. Debido a que aún no se ha desarrollado la implementación del protocolo XSTP, se mantiene la posibilidad de emplearlo como vía de notificación, pero se prevé que las notificaciones XSTP también puedan ser reemplazadas por notificaciones XSSP.

Capítulo 5. DISEÑO DE UN CLIENTE-SERVIDOR XSSP

5.1. DISEÑO INICIAL

El desarrollo del entorno de descubrimiento de servicios XSDF es muy complejo, por lo que ha sido dividido en cinco partes, correspondientes a cada protocolo (XSLP, XSRP, XSSP y XSTP), así como a la codificación binaria XBE32, para que se desarrollaran en diferentes Trabajos Fin de Grado.

En el momento del inicio de este Trabajo Fin de Grado, ya existe la implementación de los protocolos XSLP y XSRP, así como de la codificación XBE32 para el envío de mensajes XSDF. Adicionalmente, fueron proporcionadas clases de uso común entre los protocolos XSDF, que facilitan el desarrollo de agentes XSDF o la comunicación entre ellos.

Ilustración 74. Pila XSDF

Para realizar el análisis del diseño inicial, se tomó como fuente el código de XSLP y XSRP recibido al inicio del desarrollo del proyecto. Adicionalmente se han empleado las memorias de los trabajos anteriores, teniendo en cuenta que la información contenida en los mismos puede haber sufrido modificaciones respecto a la publicación de los *drafts* de XSDF.

5.1.1. PAQUETE XBE32

XBE32 [20] es una codificación jerárquica similar a XML, con la diferencia de que XBE32 es una codificación binaria. Por ello, al emplear XBE32 como codificación para los mensajes de los protocolos XSDF, se obtiene una mayor eficiencia de la red al reducir el tamaño de los mensajes.

El código recibido de XBE32 provee de los elementos necesarios para el envío y recepción de mensajes de XSDF de forma totalmente transparente a los agentes. Además, proporciona herramientas para importar y exportar ficheros XML.

Para funcionar correctamente, la capa XBE32 requiere que todos los elementos que puedan transmitirse por la red posean un identificador relacionado con el tipo de dato del elemento. Dichos identificadores se agrupan por protocolos en diccionarios para simplificar su procesamiento y comprensión por parte del administrador.

5.1.2. PAQUETE XSDF COMMON

El conjunto de clases que conforma el paquete XSDFCommon proporcionan las funcionalidades básicas para el desarrollo de los protocolos XSDF. Las clases más importantes que incluye este paquete son la implementación de un cliente y un servidor XSDF genéricos que servirán de base para los clientes y servidores específicos de cada protocolo.

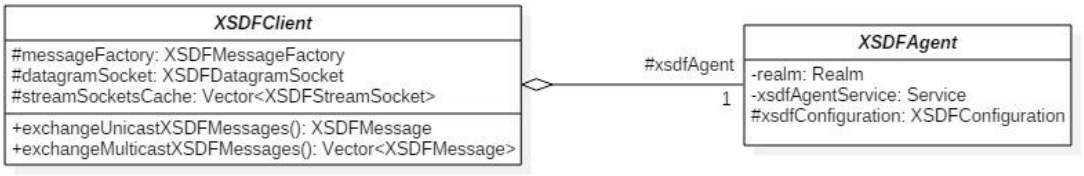

Ilustración 75. XSDFClient

La clase abstracta "*XSDFClient*" proporciona los métodos necesarios para el intercambio de mensajes tanto *unicast* como *multicast*. Un "*XSDFClient*" posee un conjunto de interfaces de comunicación tanto UDP como TCP, siendo obligatoria la interfaz de comunicación UDP y opcionales las conexiones TCP, tal y como requieren los clientes de todos los protocolos XSDF. La clase "*XSDFAgent*" permite la identificación del cliente como agente, proporcionándole información como su UUID, el reino al que pertenece, y la configuración que se le ha proporcionado.

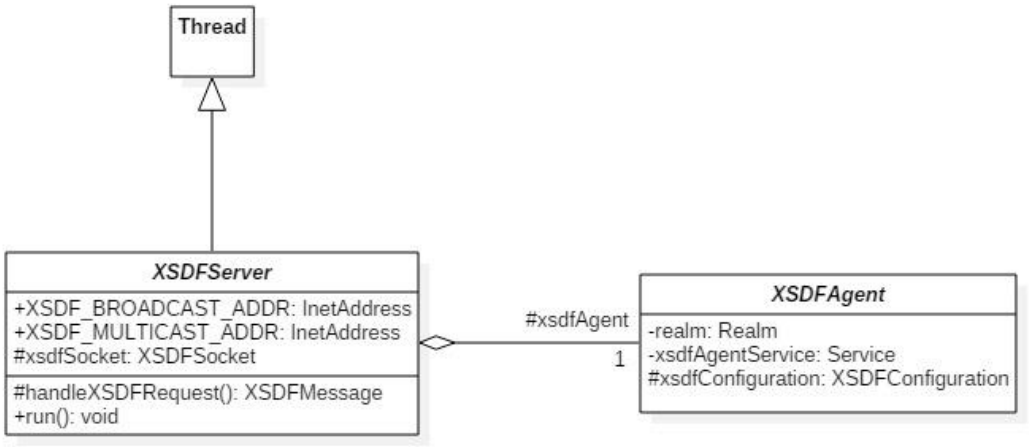

Ilustración 76. XSDFServer

La clase abstracta "*XSDFServer*" proporciona la funcionalidad de recepción asíncrona de mensajes. Mediante el método "*handleXSDFRequest*" se permite procesar los mensajes recibidos procedentes de un cliente. De igual modo al cliente, la clase "*XSDFAgent*" permite la identificación del servidor como agente, proporcionándole información como su UUID, el reino al que pertenece, y la configuración que se le ha proporcionado.

5.1.3. PAQUETE XSLP

El protocolo XSLP [4] es empleado por los UAs para obtener información de los DAs y SAs. Entre las funcionalidades que ofrece el cliente XSLP, las que se utilizarán principalmente para el desarrollo de XSSP son el descubrimiento de DAs mediante consultas multicast, y la configuración inicial del reino al que pertenece el UA.

5.1.4. PAQUETE XSRP

El protocolo XSRP [5] es empleado por los SAs para registrar sus servicios en los DAs para que las consultas XSLP vayan dirigidas a un repositorio centralizado en lugar de contactar con todos los SAs mediante *unicast*. XSRP está estrechamente relacionado con XSSP ya que cuando se registre, modifique o elimine un servicio en el DA, éste deberá notificar a los agentes XSDF que se hayan suscrito a ese servicio.

5.1.5. AGENTES XSDF

En el código recibido también se encuentran las implementaciones de los diferentes agentes que conforman XSDF, y cuyas APIs serán utilizadas para la interacción con los agentes.

5.1.5.1. AGENTE DE USUARIO

Un UA es utilizado por una aplicación de usuario para descubrir los servicios existentes en la red que cumplen las necesidades de la aplicación. Un UA posee un cliente XSLP para realizar sus transacciones.

La implementación existente divide al UA en dos entidades: “UAClient” y “UAServer”. El “UAClient” es empleado para consultas puntuales, ya que su principal ciclo de vida consiste en su creación, el descubrimiento puntual de servicios, y eliminación. Este ciclo no permite a la red trabajar de forma eficiente ya que, si varias aplicaciones del usuario necesitan un mismo servicio (por ejemplo, los DAs disponibles), deberían consultar y recibir información idéntica múltiples veces.

Por esta razón se crea el “UAServer”, una implementación en forma de demonio del UA que no detiene su ejecución y ofrece una caché local al usuario. De este modo, los diferentes “UAClient” que se ejecuten en la misma máquina pueden solicitar la información al “UAServer”, quien se encargará de devolver la información, ya sea mediante la caché local o realizando consultas XSLP a la red a modo de proxy del “UAClient”.

Además, el “UAServer” escuchará los mensajes XSLP que contengan operaciones “*ServiceAdvertisement*”, como los enviados por los DAs para anunciar su presencia en la red.

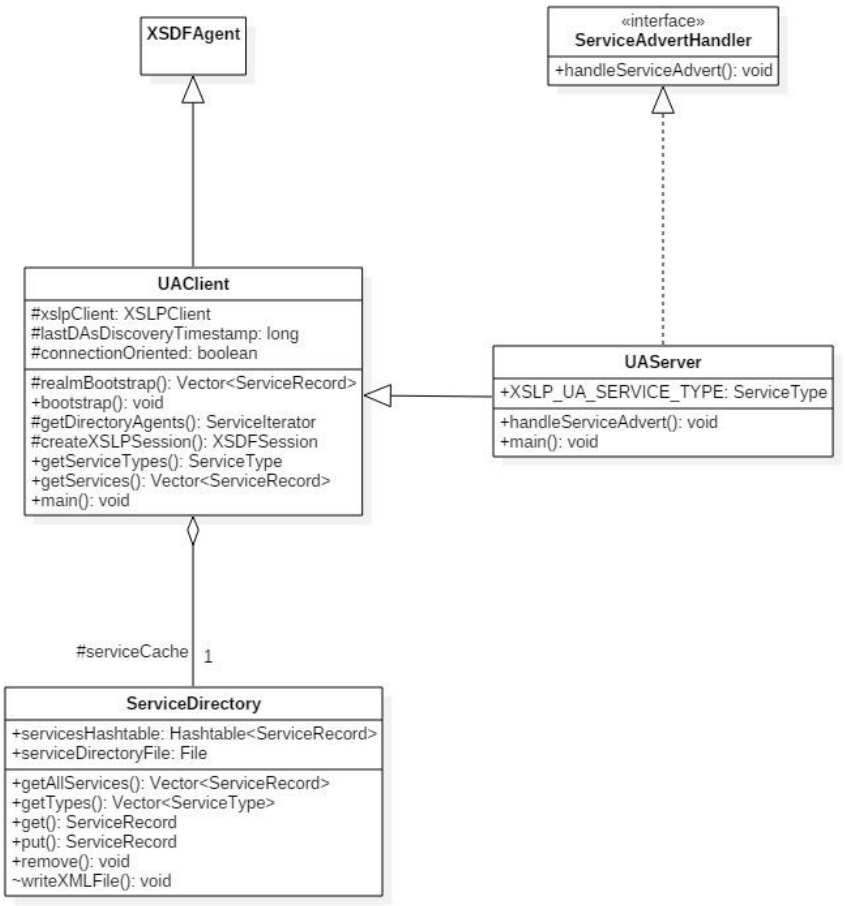

Ilustración 77. Diagrama de clases inicial de UA

Como se observa en el diagrama, el agente "UAClient" posee únicamente un cliente XSLP ya que, en principio, no necesita soportar otros protocolos. Además, proporciona todos los métodos necesarios para realizar las operaciones XSLP. El "UAServer" extiende al "UAClient" y proporciona un método para procesar los anuncios de servicios recibidos.

La clase "*ServiceDirectory*" disponible en el atributo "serviceCache" del "UAClient", almacena los servicios que ha descubierto mediante las solicitudes XSLP previas.

Para el desarrollo de aplicaciones que empleen XSDF para el descubrimiento de servicios, la clase "*UserAgent*" proporciona un API con una serie de funciones que se corresponden con las operaciones XSLP. De este modo, el funcionamiento a nivel de protocolo es completamente transparente para el usuario.

UserAgent
`connect (): void`
`disconnect (): void`
`getRealm (): Realm`
`getServiceTypes (): ServiceType []`
`getServiceTypes ( Realm realm ): ServiceType []`
`getServices ( ServiceType type ): Iterator<Service>`

`getServices ( Realm realm, ServiceType type ): Iterator<Service>`
`getFullServices ( ServiceType type ): Iterator<Service>`
`getFullServices ( Realm realm, ServiceType type ): Iterator<Service>`
`getFullService ( ServiceType type, UUID id ): Service`
`getFullService ( Realm realm, ServiceType type, UUID id ): Service`
`getXSDFConfiguration (): XSDFConfiguration`

Ilustración 78. API inicial UserAgent

Las funciones “connect()” y “disconnect()” permiten elegir si se quiere utilizar UDP o TCP para el intercambio de mensajes, siendo UDP el protocolo por defecto. La función “getRealm()” permite identificar el reino al que pertenece este agente. Las funciones “getServices()” descubren servicios de un tipo y devuelven la información del servicio sin la parte adicional. Las funciones “getFullServices()” descubren servicios de un tipo y devuelven la información completa. Las funciones “getFullService()” preguntan por un servicio concreto y devuelven su información completa. La función “getXSDFConfiguration()” devuelve un objeto que contiene la configuración del agente en ese momento.

5.1.5.2. AGENTE DE SERVICIO

Un SA es utilizado por servicios para permitir ser descubiertos por los UAs. Deben poder ser consultados directamente por los UAs mediante un servidor XSLP, y pueden registrar la información de sus servicios en los DAs mediante un cliente XSRP.

Al igual que en el caso del UA, la implementación existente divide al SA en dos entidades: “SAClient” y “SAServer”. El “SAClient” es empleado para realizar operaciones XSRP puntuales, ya que su principal ciclo de vida consiste en su creación, la comunicación con el DA para registrar o actualizar un registro, y finalización. Este ciclo no permite que el SA ofrezca sus servicios a los UAs, ya que debería poder ser descubierto por un UA en cualquier momento, y debería actualizar periódicamente los registros en el DA. Por esta razón se crea el “SAServer”, una implementación de servidor XSLP en forma de demonio del SA que no detiene su ejecución.

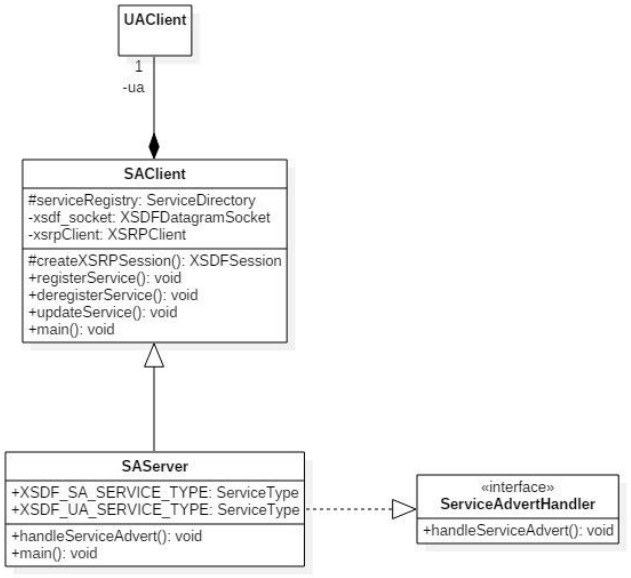

Ilustración 79. Diagrama de clases inicial de SA

Como puede observarse en el diagrama, el "SAClient" posee únicamente un cliente XSRP ya que no necesita implementar otros protocolos. Internamente cuenta con una instancia de "UAClient" que le permitirá descubrir a los DAs en caso de existir, así como descubrir el reino al que pertenece. Además, proporciona las funciones necesarias para ofrecer una funcionalidad completa de XSRP. La clase "SAServer" extiende a "SAClient" e implementa un servidor XSLP para permitir el descubrimiento de los servicios registrados localmente, así como para procesar los anuncios de servicios recibidos.

Para el desarrollo de aplicaciones que empleen XSRP para registrar servicios, la clase "*ServiceAgent*" proporciona un API con una serie de funciones de alto nivel para registrar y eliminar servicios que luego se mapean internamente a operaciones XSRP. De este modo, el funcionamiento a nivel de protocolo es completamente transparente para el usuario.

ServiceAgent
`getRealm(): Realm`
`registerService ( Service srv, int life ): void`
`registerService ( Realm realm, Service srv, int life ): void`
`deregisterService ( Service srv ): void`
`deregisterService ( Realm realm, Service srv ): void`
`updateService( Service srv ): void`
`updateService (Realm realm, Service srv): void`
`getXSDFConfiguration(): XSDFConfiguration`

Ilustración 80. API inicial ServiceAgent

Las funciones "registerService()" permiten registrar un servicio en el SA local y en un DA (si existe). Las funciones "deregisterService()" permiten eliminar la información de un servicio registrado previamente. Las funciones "updateService()" permiten cambiar la información de un servicio registrado anteriormente, así como comunicar al DA que el servicio aún está disponible.

La función “getXSDFConfiguration()” devuelve un objeto que contiene la configuración del agente en ese momento.

5.1.5.3. AGENTE DE DIRECTORIO

Un DA es empleado por los UAs y los SAs como repositorio centralizado de información de servicios. Deben poder ser consultados por los UAs mediante un servidor XSLP, y deben permitir el registro y modificación de información de servicios de los SAs mediante un servidor XSRP. Adicionalmente deben anunciar su presencia en la red periódicamente mediante un cliente XSLP que permite enviar mensajes *multicast*.

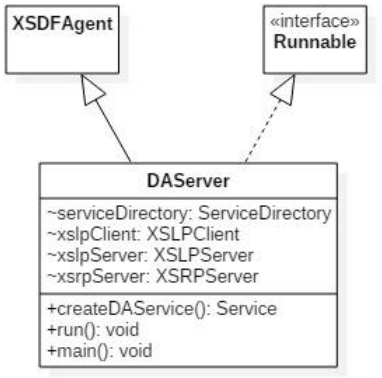

Ilustración 81. Diagrama de clases inicial de DA

Un DA es siempre un servidor ya que debe estar en constante ejecución para recibir operaciones de los UAs y SAs. Cuando recibe operaciones mediante XSRP, modifica su directorio de servicios “ServiceDirectory”, y cuando recibe operaciones XSLP responde con los servicios que han sido registrados.

5.2. DISEÑO DE XSSP

El objetivo de este proyecto es ampliar las funcionalidades ofrecidas por la implementación actual de XSDF mediante la creación de un sistema cliente-servidor XSSP que permita la suscripción a servicios. Un cliente XSSP debe poder suscribirse a los servicios de un servidor XSSP, el cual enviará una notificación cada vez que uno de los servicios que concuerda con la suscripción sea creado, modificado o eliminado.

También se deberán modificar los diferentes agentes XSDF existentes según sea conveniente para incorporar la funcionalidad de XSSP. En particular, se han modificado los Agents de Usuario y los Agentes de Directorio para añadir funcionalidad XSSP.

5.2.1. CASOS DE USO

Los siguientes casos de uso ilustran detalladamente todos los escenarios que pueden ocurrir en la implementación del protocolo XSSP. Se distinguirán como actores a los diferentes agentes existentes en XSDF: UA, SA y DA.

Tal y como muestra la siguiente ilustración, en un funcionamiento normal de XSSP se requiere la intervención de los tres tipos de agentes.

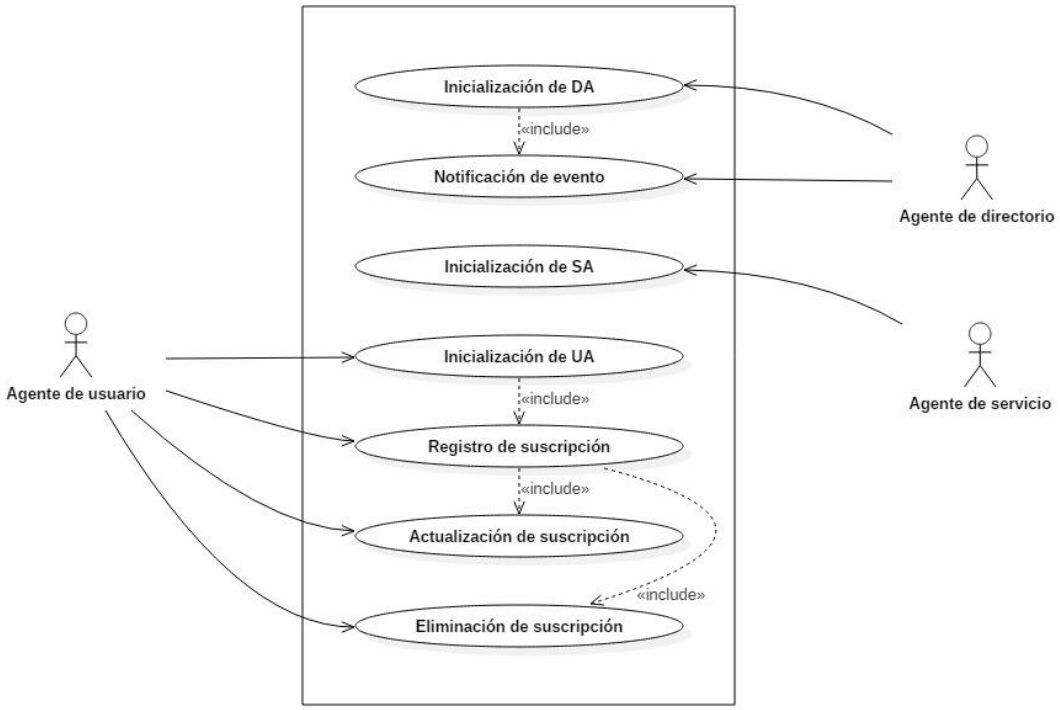

Ilustración 82. Diagrama de casos de uso de XSSP

Se considerará que la secuencia normal de los siguientes casos de uso, es aquella en la que no se ha generado ningún tipo de error, ya sea un funcionamiento previsto o no. En los escenarios alternativos se tendrán en cuenta las diferentes posibilidades de ejecución.

Nombre	Inicialización de DA.	
Descripción	Inicialización del Agente de Directorio para que actúe como repositorio central de información de servicio del ámbito al que pertenece.	
Agentes	DA.	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	Lectura de los parámetros especificados en los ficheros de configuración.
	2	Inicialización del servidor XSLP que permitirá responder con la información de servicio solicitada por los mensajes XSLP recibidos.
	3	Inicialización del cliente XSLP que permitirá anunciar su presencia en la red de forma periódica.

	4	Inicialización del servidor XSRP que permitirá recibir información de servicio y registrarla para poder actuar como repositorio centralizado.
	5	Inicialización del servidor XSSP que permitirá a otros agentes XSDF suscribirse a los cambios en la información de servicio que se produzcan en el repositorio.
	6	Envío de un mensaje "*Service Advertisement*" para anunciarse a los agentes XSDF que pueda haber activos en ese momento.
Postcondición	El DA se encuentra inicializado y sin servicios registrados excepto sí mismo.	

Tabla 7. Caso de uso: Inicialización de DA.

Nombre	Inicialización de SA.	
Descripción	Inicialización del Agente de Servicio para que gestione la información de los servicios que tiene asignados. Además, en caso de haber al menos un DA en el ámbito, registra la información.	
Agentes	SA.	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	Lectura de los parámetros especificados en los ficheros de configuración y carga de la información de los servicios estáticos que el SA tiene asociados.
	2	Localización de un DA del ámbito mediante configuración estática previa, anuncios previos o peticiones XSLP *multicast*.
	3	Inicialización del cliente XSRP que permitirá el registro de la información de servicio en el DA localizado.
	4	Inicialización del servidor XSLP que permitirá responder con la información de servicio requerida a los mensajes XSLP recibidos.
Postcondición	El SA se encuentra inicializado y el DA, en caso de haber sido desplegado, tiene servicios registrados.	
Excepciones	**Paso**	**Acción**
	2	En caso de no encontrase ningún DA activo, en el paso siguiente se inicializará el cliente XSRP, pero no se enviará información de servicio hasta localizar un DA mediante la recepción de un mensaje "*ServiceAdvertisement*".

Tabla 8. Caso de uso: Inicialización de SA.

Nombre	Inicialización de UA.	
Descripción	Inicialización del Agente de Usuario para que pueda hacer uso del entorno de descubrimiento de servicios.	
Agentes	UA.	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	Lectura de los parámetros especificados en los ficheros de configuración y carga de la información de los servicios que el SA tiene asociados.
	2	Localización de un DA del ámbito mediante configuración estática previa, anuncios previos o peticiones XSLP *multicast*.
	3	Inicialización del cliente XSLP que se empleará para realizar consultas sobre la información de los servicios.
	4	Inicialización del cliente XSSP que permitirá la suscripción a cambios en la información de los servicios.
Postcondición	El UA se encuentra inicializado.	
Excepciones	**Paso**	**Acción**
	2	En caso de no encontrarse ningún DA, las consultas XSLP se realizarán mediante *multicast* a los SAs que gestionan los servicios del ámbito.

Tabla 9. Caso de uso: Inicialización de UA.

Nombre	Creación de suscripción.	
Descripción	Un cliente XSSP se suscribe a un tipo de servicio o a un UUID en un servidor XSSP.	
Agentes	Cliente XSSP (UA) y servidor XSSP (DA).	
Precondición	Tanto el cliente como el servidor han sido inicializados previamente.	
Secuencia normal	**Paso**	**Acción**
	1	Cliente: Generación y envío de un mensaje XSSP que contiene la operación de suscripción.
	2	Servidor: Recepción del mensaje XSSP.
	3	Servidor: Si el elemento "*Subscription Target*" incluye algún ámbito, comprobar que aparezca también en la cabecera.
	4	Servidor: Comprobación de que no existe una suscripción con el mismo ID de notificación.

	5	Servidor: Comprobación de si se soporta el protocolo de notificación requerido en el elemento "*Service Information*".
	6	Servidor: Comprobación del atributo "*Global Subscription*".
	7	Servidor: Registro de la suscripción a todos los ámbitos especificados y almacenamiento de los datos sobre el servicio de notificación.
	8	Servidor: definición de la variable "*LAST_UPDATE_TIMESTAMP*" con el momento en que se produce la suscripción y de un temporizador "*MAX_LIFE_TIMER*" asociado al nuevo registro.
	9	Servidor: Elección de un intervalo de actualización y envío de un mensaje "*Subscribe Service Ack*".
	10	Cliente: Planificar la actualización de la suscripción.
Postcondición	La suscripción del cliente XSSP se ha registrado con éxito en el servidor XSSP.	
Excepciones	**Paso**	**Acción**
	3	En caso de que los ámbitos contenidos en el elemento "*Subscription Target*" no coincidan con los existentes en la cabecera, se detiene el caso de uso sin enviar respuesta.
	4	En caso de que exista una suscripción con el mismo ID, se envía una respuesta "*Error*" de tipo "*SUBSCRIPTION_COLLISION*" y detiene el caso de uso.
	5	En caso de no soportar el protocolo de notificación requerido, se envía una respuesta "Error" de tipo "*UNSUPPORTED_PROTOCOL*" y detiene el caso de uso.
	6	En caso de que se solicite una suscripción global a un servidor XSSP que no soporte suscripciones globales se envía una respuesta "Error" de tipo "*INVALID_SUBSCRIPTION*" y detiene el caso de uso.

Tabla 10. Caso de uso: Creación de suscripción.

Nombre	Actualización de suscripción.	
Descripción	Un cliente XSSP actualiza una suscripción a un tipo de servicio o un UUID en un servidor XSSP.	
Agentes	Cliente XSSP (UA) y servidor XSSP (DA).	
Precondición	Tanto el cliente como el servidor han sido inicializados previamente, y se ha creado la suscripción.	
Secuencia normal	**Paso**	**Acción**

	1	Cliente: Generación y envío de un mensaje XSSP que contiene las operaciones de actualización de la suscripción.
	2	Servidor: Recepción del mensaje XSSP.
	3	Servidor: Si el elemento "*Subscription Target*" incluye algún ámbito, comprobar que aparezca también en la cabecera.
	4	Servidor: Comprobación de que existe una suscripción con el ID de notificación especificado.
	5	Servidor: Comprobar que el atributo "*Global Subscription*" del elemento "*Subscription Information*" coincide con el registrado.
	6	Servidor: Actualización de la suscripción en todos los ámbitos especificados.
	7	Servidor: definición de la variable "*LAST_UPDATE_TIMESTAMP*" con el momento en que se produce la actualización y de un temporizador "*MAX_LIFE_TIMER*" asociado al registro actualizado.
	8	Servidor: Elección de un intervalo de actualización y envío de un mensaje "*Update Subscription Ack*".
	9	Cliente: Planificar siguiente actualización.
Postcondición	La suscripción del cliente XSSP se ha actualizado con éxito en el servidor XSSP.	
Excepciones	**Paso**	**Acción**
	3	En caso de que los ámbitos contenidos en el elemento "*Subscription Target*" no coincidan con los existentes en la cabecera, se detiene el caso de uso sin enviar respuesta.
	4	En caso de que no exista ninguna suscripción con el ID incluido en la operación, se envía una respuesta "*Error*" de tipo "*SUBSCRIPTION_NOT_FOUND*" y detiene el caso de uso.
	5	En caso de que el atributo "*Global Subscription*" sea incorrecto, se envía una respuesta "Error" de tipo "*INVALID_SUBSCRIPTION*" y detiene el caso de uso.

Tabla 11. Caso de uso: Actualización de suscripción.

Nombre	Eliminación de suscripción.
Descripción	Un cliente XSSP elimina una suscripción a un tipo de servicio o un UUID en un servidor XSSP.
Agentes	Cliente XSSP (UA) y servidor XSSP (DA).

Precondición	Tanto el cliente como el servidor han sido inicializados previamente, y se ha creado la suscripción.	
Secuencia normal	**Paso**	**Acción**
	1	Cliente: Generación y envío de un mensaje XSSP que contiene las operaciones de eliminación de la suscripción.
	2	Servidor: Recepción del mensaje XSSP.
	3	Servidor: Si el elemento "*Subscription Target*" incluye algún ámbito, comprobar que aparezca también en la cabecera.
	4	Servidor: Comprobación de que existe una suscripción con el ID de notificación especificado.
	5	Servidor: Eliminación de la suscripción en todos los ámbitos especificados.
	6	Servidor: Elección de un intervalo de actualización y envío de un mensaje "*Unsubscribe Service Ack*".
Postcondición	La suscripción del cliente XSSP se ha eliminado con éxito en el servidor XSSP.	
Excepciones	**Paso**	**Acción**
	3	En caso de que los ámbitos contenidos en el elemento "*Subscription Target*" no coincidan con los existentes en la cabecera, se detiene el caso de uso sin enviar respuesta.
	4	En caso de que no exista ninguna suscripción con el ID incluido en la operación, se envía una respuesta "*Error*" de tipo "*SUBSCRIPTION_NOT_FOUND*" y detiene el caso de uso.

Tabla 12. Caso de uso: Eliminación de suscripción.

Nombre	Envío de notificación	
Descripción	Un servidor XSSP envía una notificación de evento a los clientes XSSP suscritos.	
Agentes	Cliente XSSP (UA) y servidor XSSP (DA).	
Precondición	Tanto el cliente como el servidor han sido inicializados previamente, y se ha creado la suscripción.	
Secuencia normal	**Paso**	**Acción**
	1	Servidor: Se produce un cambio en la información de servicio almacenada en el "*ServiceDirectory*".
	2	Servidor: Comprobación de si el evento coincide con los criterios establecidos por la suscripción del cliente XSSP.

	3	Servidor: Generación y envío de un mensaje XSSP con una operación "*Service Event*".
	4	Cliente: Recepción del mensaje XSSP.
	5	Cliente: Modificación de la base de datos de servicios, y notificación a las aplicaciones o al usuario.
Postcondición	El cliente XSSP es informado de la creación, modificación o eliminación de un servicio.	
Excepciones	**Paso**	**Acción**
	2	Si el evento producido no se corresponde con los criterios de ninguna suscripción, se interrumpe el caso de uso.

Tabla 13. Caso de uso: Envío de notificación

5.3. DISEÑO DEL API DEL USER AGENT

El API del UserAgent ha sido modificado para incluir las dos operaciones XSSP que puede ejecutar el UA: "*subscribeService()*" y "*unsubscribeService()*". El objeto devuelto al crear una suscripción, "*ServiceSubscription*" almacena todos los datos de la subscripción, facilitando al cliente la interacción con XSSP. Para eliminar una suscripción, únicamente será necesario indicar el "*ServiceSubscription*" devuelto en el momento de crear la suscripción que se quiere borrar.

UserAgent
`subscribeService (ServiceType serviceType, Vector<UUID> ids, ServiceListener listener ): ServiceSubscription`
`subscribeService ( Realm realm, ServiceType serviceType, Vector<UUID> ids, ServiceListener listener ): ServiceSubscription`
`unsubscribeService ( ServiceSubscription subscription ): void`

Tabla 14. API modificado UserAgent

No es necesario que la aplicación renueve la suscripción explícitamente, la implementación del cliente XSSP se encarga de renovarlas automáticamente cuando expire el temporizador.

Capítulo 6. IMPLEMENTACIÓN DE UN CLIENTE-SERVIDOR XSSP

En los siguientes apartados se describirá detalladamente la implementación que se ha realizado del protocolo XSSP, sus agentes y sus operaciones, basados en los casos de uso descritos en el capítulo anterior.

6.1. MENSAJES XSSP

En este apartado se describirá la implementación de los mensajes pertenecientes al protocolo XSSP. Para información detallada de la información que contiene cada elemento se recomienda la lectura previa del apartado 4.1.

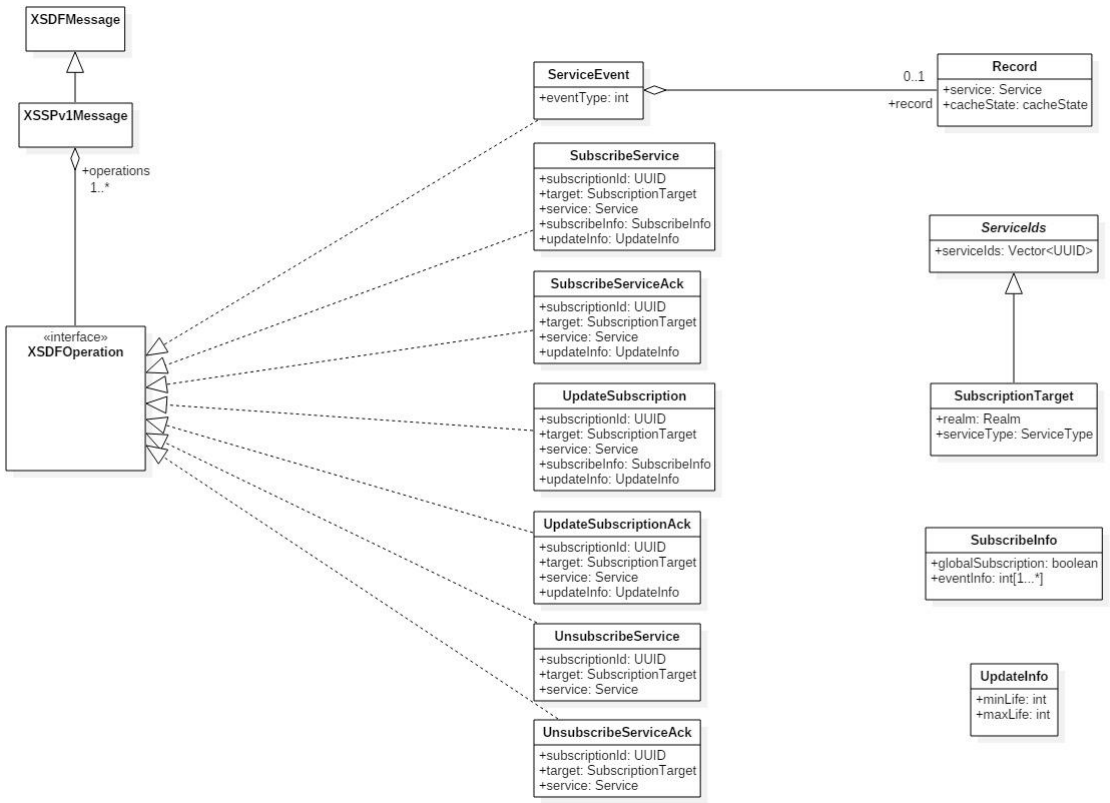

Ilustración 83. Implementación de mensajes XSSP

El contenedor para las operaciones XSSP es una instancia de la clase "*XSSPv1Message*", derivada de la clase "*XSDFMessage*", definida en XSDFCommon. Un objeto "*XSSPv1Message*" contiene una o más operaciones XSSP. Las operaciones son instancias de las clases "*SubscribeService*", "*SubscribeServiceAck*", "*UpdateSubscription*", "*UpdateSubscriptionAck*", "*UnsubscribeService*", "*UnsubscribeServiceAck*" y "*ServiceEvent*", que deben implementar la interfaz "*XSDFOperation*" incluida en XSDFCommon.

Todos los elementos extienden de la clase "*XBE32SerializableImpl*" incluida en XBE32, que permite su codificación con XBE32 para ser transmitidos por la red.

6.2. CLIENTE XSSP

El cliente XSSP provee al agente XSDF que lo ejecuta, la posibilidad de crear y gestionar suscripciones en un servidor XSSP.

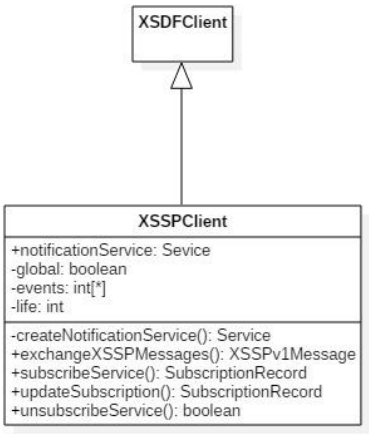

Ilustración 84. Implementación de cliente XSSP

La clase "*XSSPClient*" hereda de la implementación genérica de un cliente, "*XSDFClient*", descrita en el apartado 5.1.2 de esta memoria. Los atributos privados "global", "events" y "life" son parámetros para las operaciones XSSP. Toman valores por defecto mediante el constructor y pueden ser modificados en caso de que el usuario lo requiera. El atributo "notificationService" define el servicio de notificación que se incluirá en las operaciones XSSP para identificar y localizar al suscriptor.

Mediante el método "*exchangeXSSPMessages*" el cliente puede enviar un mensaje a un servidor XSSP y recibir una respuesta, ya sea un ACK de las operaciones enviadas o un mensaje de error. Mediante los métodos "*subscribeService()*", "*updateSubscription()*" y "unsubscribeService()", el cliente crea un mensaje XSSP con una operación y llama al método "*exchangeXSSPMessages()*" para recibir una respuesta del servidor y procesarla.

6.3. SERVIDOR XSSP

El servidor XSSP gestiona las suscripciones de los clientes mediante la recepción de operaciones XSSP y el envío de respuestas; y notifica de los eventos que se produzcan en la base de datos de servicios del agente a los clientes suscritos al tipo o UUID del servicio modificado.

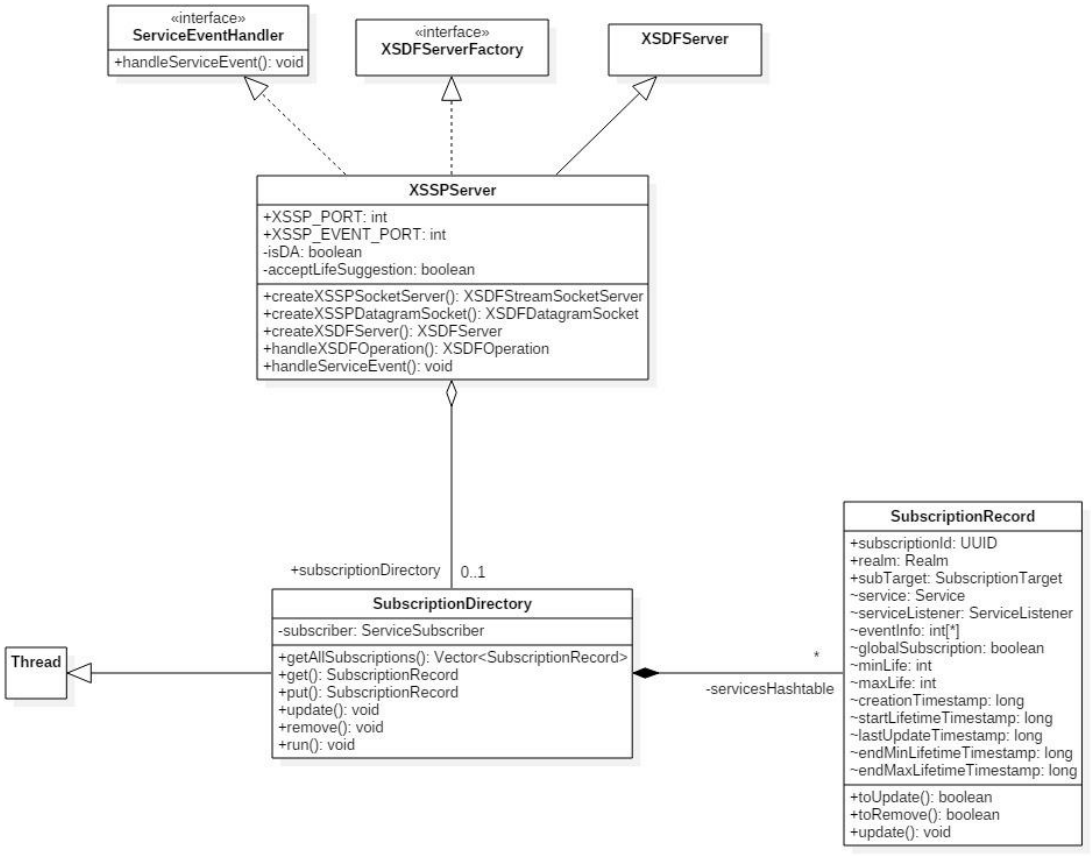

Ilustración 85. Implementación de servidor XSSP

La clase "*XSSPServer*" es la implementación de un servidor XSSP. Hereda de la implementación genérica de un servidor, "*XSDFServer*", descrita en el apartado 5.1.2 de esta memoria. Por facilidad de uso, XSSP emplea dos puertos diferentes: uno para el intercambio de operaciones de gestión de suscripciones, y el otro para la recepción de notificaciones. Estos puertos se definen en las variables estáticas "*XSSP_PORT*" y "*XSSP_EVENT_PORT*" respectivamente.

El atributo "*isDA*" indica si el servidor XSSP se está ejecutando en un DA, y por tanto deberá aceptar suscripciones globales, o si por el contrario se está ejecutando en un SA y no debería aceptar suscripciones globales. El atributo "*acceptLifeSuggestion*" le indica al servidor si debería tener en cuenta o no los valores "*maxLife*" y "*minLife*" que envíe un cliente XSSP en las operaciones "*Subscribe Service*" y "*Update Subscription*".

El principal método de esta clase es "*handleXSDFOperation()*", que es invocado al recibir un mensaje con operaciones XSSP. Este método procesa la operación, realiza los cambios necesarios en la base de datos de suscripciones y elabora la operación de respuesta. Mediante los métodos "*createXSSPSocketServe()r*" y "*createXSSPDatagramSocket()*" se crean los sockets necesarios para la configuración de red del servidor XSSP. Con el método "*createXSDFServer()*" se implementa la interfaz "*XSDFServerFactory*", y simplemente crea una nueva instancia de "XSSPServer".

El método "*handleServiceEvent()*" implementa la interfaz "*ServiceEventHandler*" y permite al servidor recibir los cambios que se han producido en la base de datos de servicios para poder notificar a los suscriptores, ya sean producidos los cambios por la recepción de una operación XSRP (en el caso de un DA) o por el registro local de un nuevo servicio (en el caso de un SA).

El servidor XSSP lleva asociada un repositorio de suscripciones denominada "*SubscriptionDirectory*". En este objeto se almacenan las suscripciones de los clientes, caracterizadas por la clase "*SubscriptionRecord*", mediante una tabla hash. El uso de una tabla hash agiliza el proceso de búsqueda de suscripciones usando como índice el identificador de la suscripción.

Tanto un "ServiceSubscriber" (explicado en el siguiente punto) como un servidor XSSP poseen un objeto "*SubscriptionDirectory*", por lo que en el caso del servidor XSSP este atributo deberá ser null. Usando el método "*getAllSubscriptions()*" se consigue una lista con todas las suscripciones. Mediante el método "*get()*" se puede conseguir una suscripción específica. Mediante los métodos "*put()*", "*update()*" y "*remove()*" se puede crear, actualizar o eliminar una suscripción respectivamente.

Una instancia de "*SubscriptionDirectory*" ejecuta un proceso que comprueba si hay que actualizar la suscripción (solo en el caso de que esté asociado a un "*ServiceSubscriber*") o si hay que eliminarla, basándose en las marcas temporales de las suscripciones o los métodos "*toUpdate()*" y "toRemove()".

6.4. SERVICE SUBSCRIBER

A diferencia de los protocolos XSLP y XSRP, los agentes XSDF no manipulan directamente un cliente XSSP, sino que poseerán una instancia de la clase "*ServiceSubscriber*" que realizará de forma automática algoritmos de actualización de suscripciones, creación de sesiones o re-suscripción entre otros.

Se decidió la creación de este agente XSSP para simplificar al máximo las funciones de los agentes XSDF existentes, agrupando toda la funcionalidad XSSP en una clase.

El *Service Subscriber* gestiona las suscripciones en el servidor XSSP mediante llamadas al cliente XSSP; y recibe las notificaciones, modifica la base de datos de servicios y notifica al agente que realizó la suscripción.

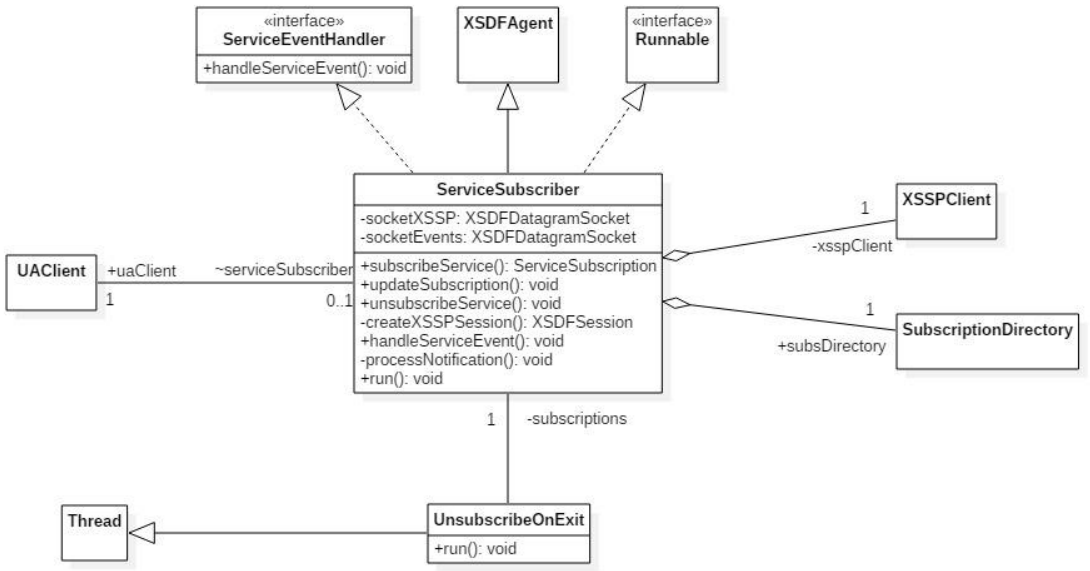

Ilustración 86. Implementación de ServiceSubscriber

El objeto "*ServiceSubscriber*" extiende de la clase "*XSDFAgent*" que permite la identificación de los agentes. Posee una instancia de "*XSSPClient*" para solicitar el envío de mensajes a un servidor, y un objeto "*SubscriptionDirectory*" en el que almacena las suscripciones locales que debe gestionar. También puede emplear una instancia de "*UAClient*" para descubrir los DAs existentes en la red.

Los atributos "*socketXSSP*" y "*socketEvents*" contienen los sockets necesarios para la comunicación del agente.

Mediante los métodos "*subscribeService()*", "*updateSubscription()*" y "*unsubscribeService()*" permite solicitar al cliente XSSP el envío de operaciones al servidor. El método "*createXSSPSession*" crea una sesión para el envío de los mensajes.

La ejecución del proceso del "*ServiceSubscriber*" permite recibir mensajes de notificación de un servidor XSSP. Cuando esto sucede, ejecuta el método "*processNotification()*" que modifica la base de datos de servicios.

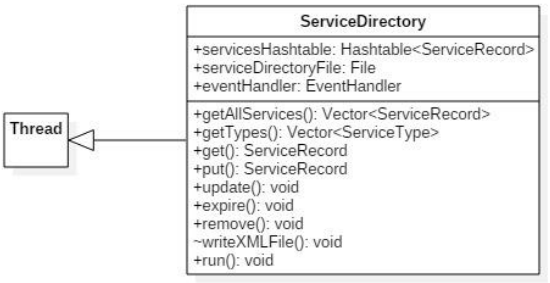

Ilustración 87. Implementación ServiceDirectory

La base de datos de servicios, la clase "*ServiceDirectory*" expuesta en el apartado 5.1.5.1 de esta memoria ha sido modificado para que al alterar cualquier servicio notifique a un objeto que implemente la interfaz "*ServiceEventHandler*". Al llamar a la función "*handleServiceEvent()*" del *Service Subscriber*, se buscará en la base de datos de suscripciones aquellas cuyo criterio coincida con el servicio notificado, y se les enviará una notificación.

6.5. AGENTES XSDF

Se ha modificado la implementación existente de los agentes XSDF (UA y DA) explicados en el apartado 5.1.5 de la memoria.

6.5.1. MODIFICIACIÓN DEL AGENTE DE USUARIO

Dado que se empleará un objeto de tipo "*ServiceSubscriber*" para realizar suscripciones, la clase "*UAClient*" ha sufrido pocas modificaciones con respecto al original.

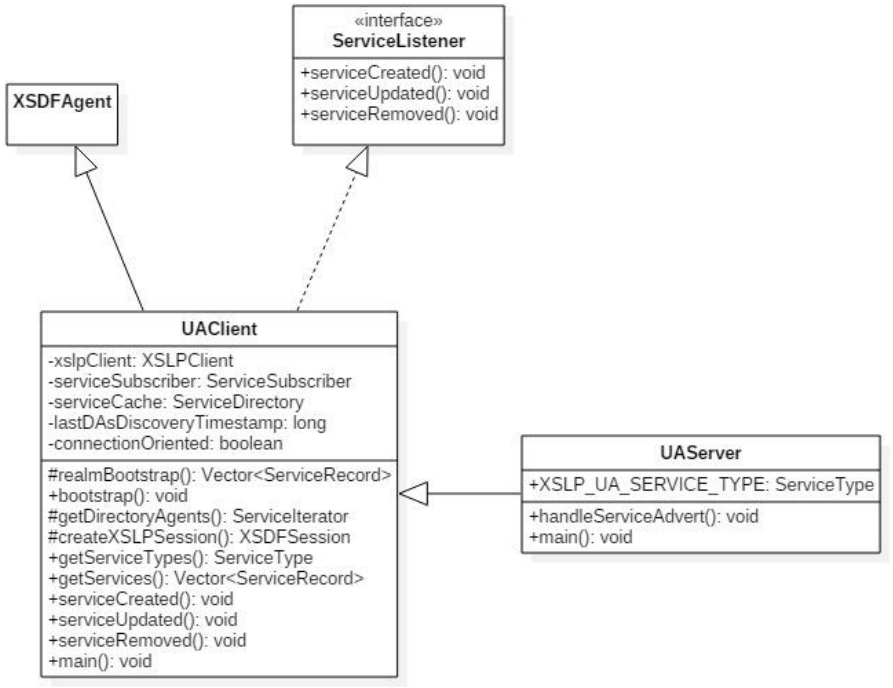

Ilustración 88. Implementación de UA

El atributo "*serviceSubscriber*" contiene una instancia del agente XSSP definido previamente. Si el UA no requiere ninguna suscripción, no crea el "*ServiceSubscriber*" para ahorrar recursos.

Los nuevos métodos "*serviceCreated()*", "*serviceUpdated()*" y "*serviceRemoved()*" resultan de la implementación de la interfaz "*ServiceListener*" que debe implementar la aplicación que usa el API de *UserAgent*. De este modo, el UA podrá recibir las notificaciones relacionadas con sus suscripciones.

6.5.2. MODIFICIACIÓN DEL AGENTE DE DIRECTORIO

La clase "*DAServer*" ha sido modificada para que también ofrezca funcionalidades de servidor XSSP.

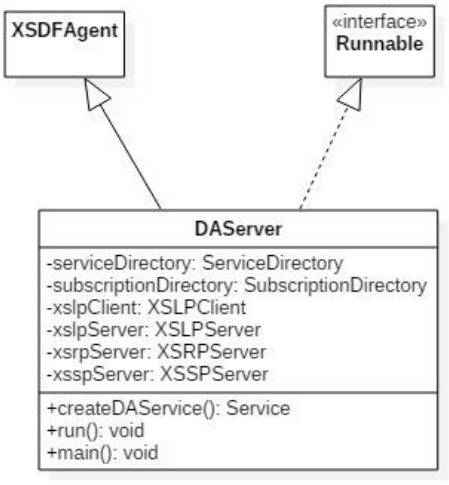

Ilustración 89. Implementación de DA

El correcto funcionamiento de los agentes XSSP en los DA posibilitará, junto con la futura implementación que se realizará de XSTP, la sincronización de múltiples DAs, haciendo escalable al entorno de XSDF.

El servidor XSSP se crea y ejecuta en el constructor de la clase "DAServer" junto con los servidores XSLP y XSRP.

6.6. OPERACIONES XSSP

Para facilitar la comprensión de los diferentes elementos que conforman XSSP, se expondrán en este apartado diversos diagramas de secuencias que se corresponden con las diferentes operaciones de XSSP.

Para realizar los ejemplos se supondrá que el cliente XSSP es un UA, pero la implementación permite que sea cualquier agente XSDF, basta con que cree una instancia de "*ServiceSubscriber*".

6.6.1. CREACIÓN DE SUSCRIPCIÓN

Para este caso se asume que una aplicación ha empleado el API de *UserAgent* para registrar una suscripción en un servidor XSSP.

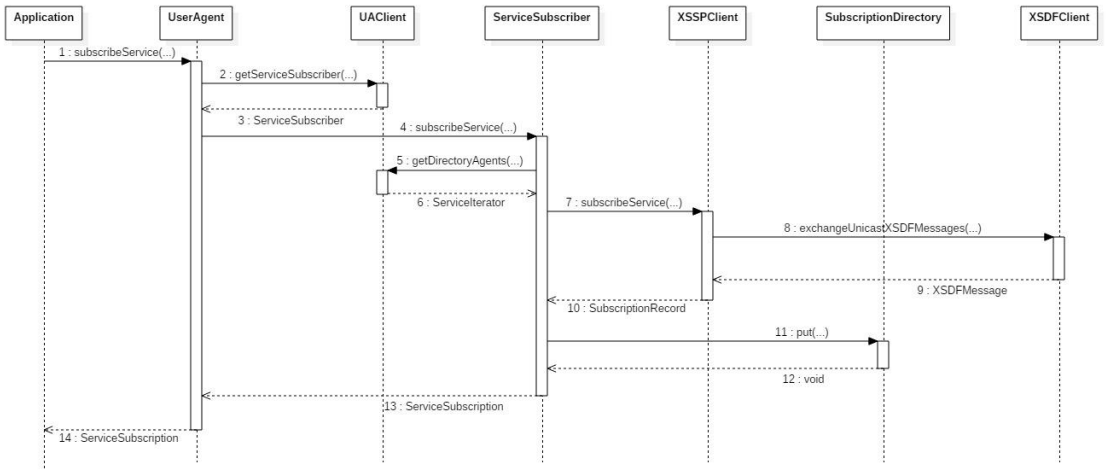

Ilustración 90. Diagrama de secuencia: Creación de suscripción.

Tras recibir la orden, el API invocará al agente "*ServiceSubscriber*", que generará una sesión y solicitará al "*XSSPClient*" que envíe una operación XSSP. El cliente XSSP envía un mensaje con la operación y recibe una respuesta, que en caso de ser un ACK, procesa en un "*SubscriptionRecord*" y se lo devolverá al "*ServiceSubscriber*". Éste registrará la nueva suscripción en su "*SubscriptionDirectory*" y enviará un objeto "*ServiceSubscription*" de vuelta al "*UserAgent*", que lo devolverá a la aplicación del usuario.

6.6.2. MODIFICACIÓN DE SUSCRIPCIÓN

El proceso que ejecuta una instancia de "*SubscriptionDirectory*" comprueba periódicamente todas las suscripciones y determina cuando es necesario actualizar una suscripción para que no sea eliminada en el servidor XSSP.

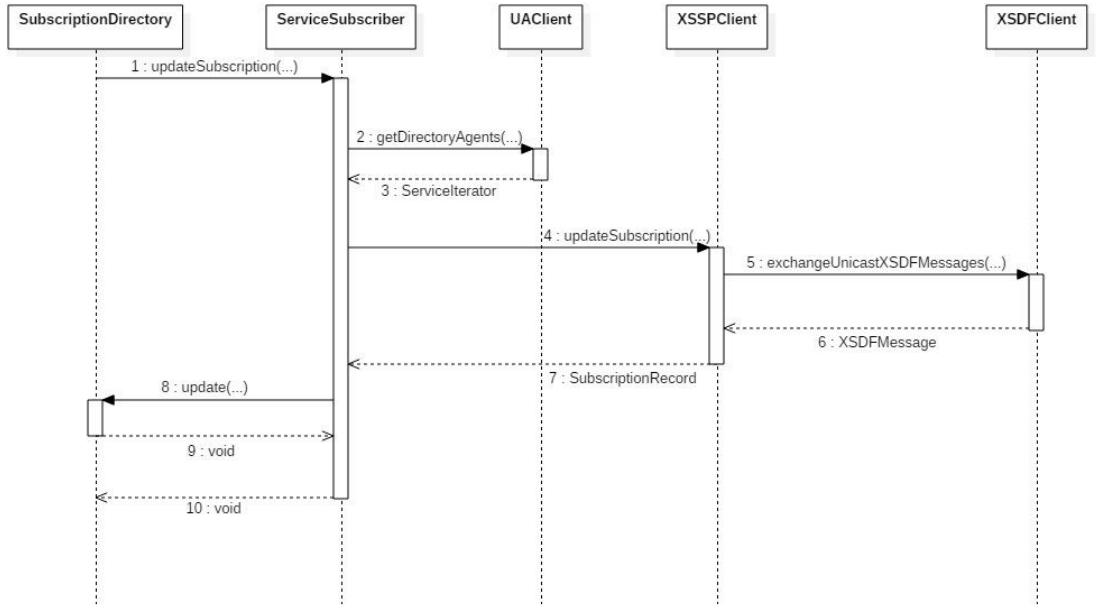

Ilustración 91. Diagrama de secuencia: Actualización de suscripción.

Cuando un "*SubscriptionDirectory*" determina que hay que actualizar una suscripción, informa al "*ServiceSubscriber*". Éste creará una sesión XSSP y solicitará al "*XSSPClient*" que envíe una operación de actualización al servidor XSSP. El cliente elabora un mensaje con la operación y envía el mensaje al servidor, recibiendo un mensaje de respuesta. Si la respuesta es un ACK, la procesa y elabora un "*SubscriptionRecord*" que devuelve al "*ServiceSubscriber*", quien actualiza la suscripción.

6.6.3. ELIMINACIÓN DE SUSCRIPCIÓN

De forma similar a la creación de una suscripción, se asume que una aplicación hace uso del API del *UserAgent* para eliminar una suscripción.

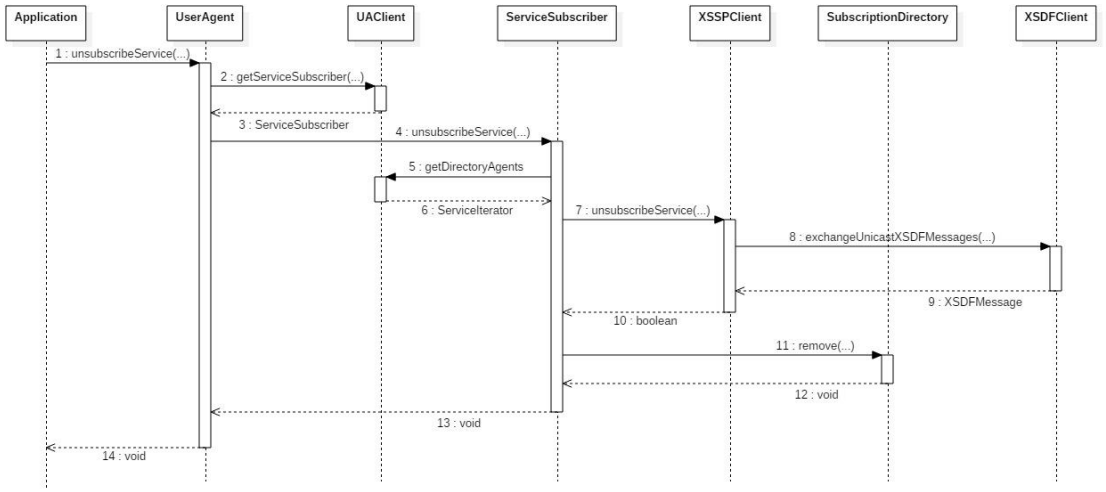

Ilustración 92. Diagrama de secuencia: Eliminación de suscripción.

Tras recibir la orden, el API invocará al agente "*ServiceSubscriber*", que generará una sesión y solicitará al "*XSSPClient*" que envíe una operación XSSP. El cliente XSSP envía un mensaje con la

operación y recibe una respuesta. En caso de ser un ACK informa al "ServiceSubscriber" de que el servidor XSSP ha eliminado la suscripción. Éste eliminara la suscripción del su "*SubscriptionDirectory*".

6.6.4. SERVIDOR XSSP

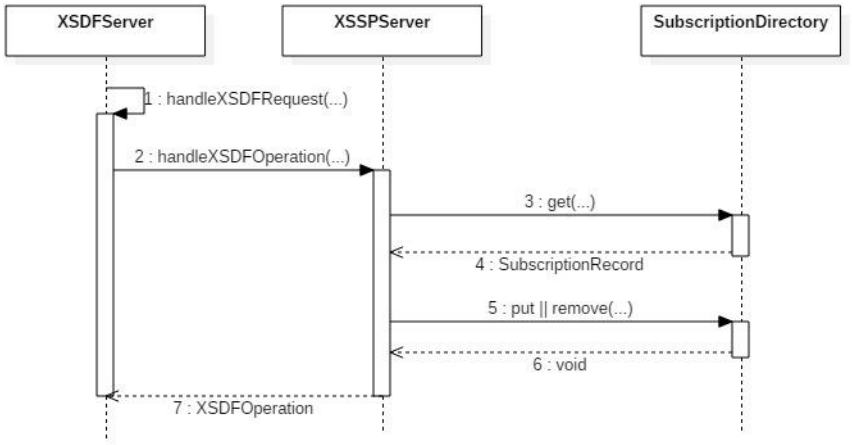

Ilustración 93. Diagrama de secuencia: mensaje recibido en servidor XSSP.

Por parte del servidor XSSP, la secuencia que sigue al recibir una operación de creación, modificación o eliminación de una suscripción es siempre el mismo. El método "*handleXSDFRequest()*" implementado en la clase "*XSDFServer*" recibe un mensaje con el socket definido para el servidor XSSP. Este método divide en operaciones el mensaje y llama una vez por cada operación al método "*handleXSDFOperation()*" implementado en la clase "*XSSPServer*". En este momento se analiza la operación y se comprueba si es necesario mandar algún error. Si no es necesario, se ejecutará una operación "*put()*" o "*remove()*" en la base de datos de suscriptores y se devolverá una operación de respuesta.

6.6.5. NOTIFICACIÓN DE EVENTO

El propósito principal de XSSP es lograr que un servidor XSSP informe a un cliente XSSP de que se han producido cambios en la información de algún servicio relacionado con su suscripción. En este caso, lógicamente el comportamiento por parte del servidor es diferente al expuesto al principio del apartado.

6.6.5.1. NOTIFICACIÓN DE EVENTO EN EL SERVIDOR

El servidor XSSP debe notificar los cambios producidos ya sea mediante XSRP (en caso de un DA) o mediante la alteración local de un servicio (en caso de un SA). Por ello, será la propia base de datos de servicios la que notificará que un servicio se ha modificado.

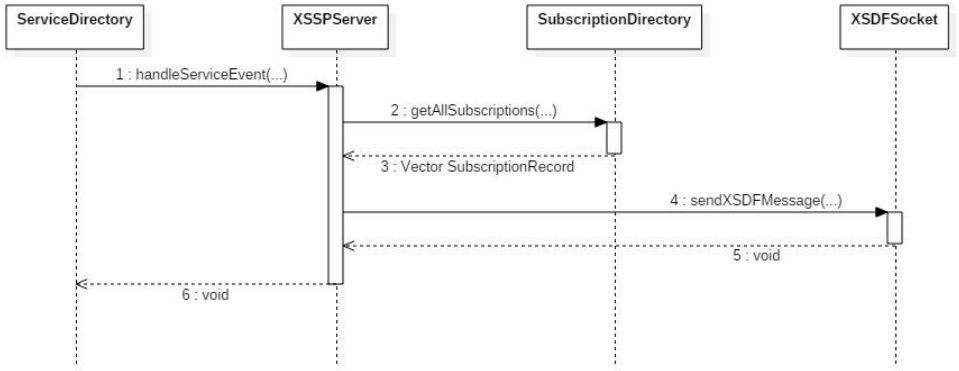

Ilustración 94. Diagrama de secuencia: Notificación de evento - servidor

Al producirse el registro, eliminación o alteración de un servicio en el "*ServiceDirectory*", se realiza una llamada al "*XSSPServer*". Éste comprueba con que suscripciones del "*SubscriptionDirectory*" se corresponde el evento que se ha producido y procede a enviar un mensaje de notificación a cada uno de ellos.

6.6.5.2. NOTIFICACIÓN DE EVENTO EN EL CLIENTE

En el lado del cliente, no será la clase "*XSSPClient*" la que reciba el mensaje, sino la clase "*ServiceSubscriber*" ya que el cliente no puede ejecutar un proceso ya que entraría en conflicto con la definición de cliente estático de la clase "*XSDFClient*".

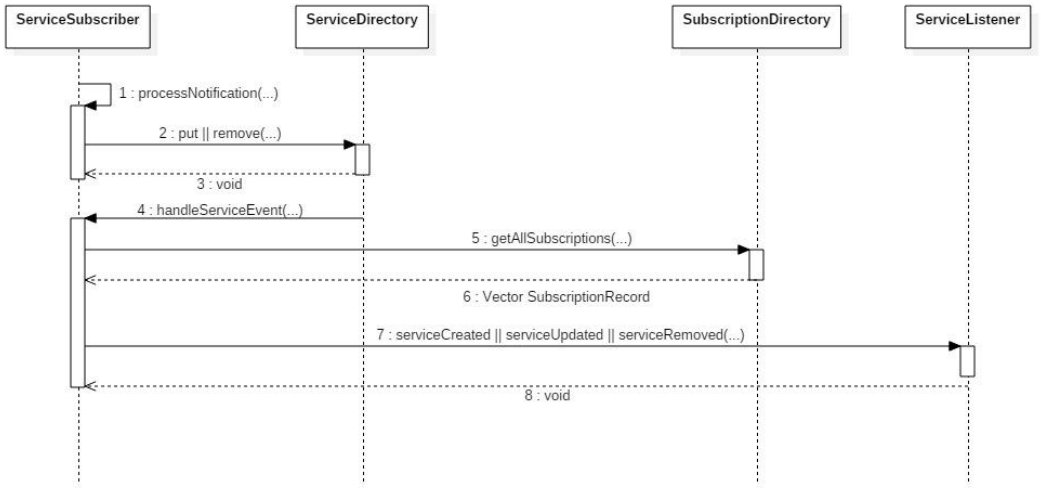

Ilustración 95. Diagrama de secuencia: Notificación de evento - cliente

Cuando durante la ejecución del proceso del "*ServiceSubscriber*" se recibe un mensaje de notificación, se llama al método "*processNotification()*". Mediante este método, la clase "*ServiceSubscriber*" realiza los cambios necesarios en el "*ServiceDirectory*".

Al producirse un cambio en la base de datos de servicios, se dispara el método "*handleServiceEvent()*" de forma similar al caso del servidor, con la diferencia de que es "*ServiceSubscriber*" quien implementa la interfaz "*ServiceEventHandler*". Finalmente, el "*ServiceSubscriber*" comprueba su "*SubscriptionDirectory*" para comprobar qué aplicación ha realizado la suscripción y la notifica con los métodos "*serviceCreated()*", "*serviceUpdated()*" o "*serviceRemoved()*" dependiendo del tipo de cambio ocurrido.

Capítulo 7. EVALUACIÓN Y PRUEBAS

Para garantizar que todos los componentes desarrollados en el ámbito de este Trabajo Fin de Grado funcionan correctamente, se han diseñado pruebas exhaustivas que comprueban el buen comportamiento de las aplicaciones.

Tras la realización de las pruebas, se concluye que los diferentes componentes desarrollados se comportan correctamente en un escenario normal, y que son también suficientemente robustos para gestionar la detección y envío de errores.

Para la realización de estas pruebas se han desplegado varias máquinas virtuales en VirtualBox, correspondientes a un DA, un SA y varios UAs, y se han conectado entre ellas mediante una red interna. No se realiza ninguna configuración estática, sino que los agentes se descubren dinámicamente empleando el protocolo XSLP, desarrollado en un Trabajo Fin de Carrera anterior [4].

7.1. CREACIÓN DE SUSCRIPCIÓN - FUNCIONAMIENTO NORMAL

Se comprueba que el UA puede crear una nueva suscripción en el DA. Para ello, primero deberá descubrir al DA mediante XSLP y enviar un mensaje XSSP con una operación "*Subscribe Service*". El DA deberá registrar la suscripción y enviar un mensaje de respuesta con una operación "*Subscribe Service Ack*".

Esta prueba se realiza tanto para la suscripción individual de un tipo de servicio o un UUID, como para la suscripción a múltiples elementos.

El resultado es correcto, todos los tipos de suscripción se pueden crear correctamente.

7.2. CREACIÓN DE SUSCRIPCIÓN - COLISIÓN DE ID

Se comprueba el envío de un error en caso de que una suscripción ya exista en el DA con un ID.

Para comprobar este funcionamiento, se registra un UA en el DA, que debe suscribirse correctamente, y después se intenta registrar otro UA con el mismo ID, el cual debe recibir una operación "*Error*" de tipo "*SUBSCRIPTION_COLLISION"*.

Para realizar esta prueba, se modifica ligeramente el cliente XSSP para evitar que genere un ID aleatorio.

El resultado es el esperado, la primera suscripción se crea correctamente, mientras que la segunda falla por utilizar el mismo ID.

7.3. CREACIÓN DE SUSCRIPCIÓN - PROTOCOLO NO SOPORTADO

Se comprueba el envío de un error en caso de que el DA no soporte el protocolo de notificación que solicita el UA.

Para comprobar este funcionamiento, se intenta registrar un UA en el DA, el cual debe recibir una operación "*Error*" de tipo "*UNSUPPORTED_PROTOCOL"*.

Para realizar esta prueba, se especifica al UA que solicite que los eventos se notifiquen mediante XSTP, protocolo aún no implementado.

El resultado es el esperado, el servidor no crea la suscripción y responde con un mensaje de error.

7.4. ACTUALIZACIÓN DE SUSCRIPCIÓN - FUNCIONAMIENTO NORMAL

Al registrar una suscripción, el UA recibe en la operación "*Subscribe Service Ack*" los valores de tiempo mínimo y máximo en los que debe actualizar la suscripción. En caso de que no se actualice en el tiempo indicado, la suscripción será eliminada.

El UA deberá por tanto enviar una operación "*Update Subscription*" entre los dos tiempos indicados y recibir una operación "*Update Subscription Ack*".

El resultado es correcto, la suscripción se actualiza adecuadamente.

7.5. ACTUALIZACIÓN DE SUSCRIPCIÓN - SUSCRIPCIÓN NO ENCONTRADA

Si el UA ha tardado demasiado tiempo en actualizar la suscripción, o envía la actualización de un servicio que no está suscrito en el DA, el UA deberá recibir una operación "*Error*" de tipo "*SUBSCRIPTION_NOT_FOUND"*.

El resultado es el esperado, al intentar actualizar una suscripción inexistente, el servidor responde con un mensaje de error.

7.6. ELIMINACIÓN DE SUSCRIPCIÓN - FUNCIONAMIENTO NORMAL

Se comprueba que el UA puede eliminar una suscripción que anteriormente creó en el DA. Para ello, deberá enviar un mensaje XSSP con una operación "*Unsubscribe Service*". El DA deberá eliminar la suscripción y enviar un mensaje de respuesta con una operación "*Unsubscribe Service Ack*".

El resultado es correcto, la suscripción se elimina del DA correctamente.

7.7. ELIMINACIÓN DE SUSCRIPCIÓN - SUSCRIPCIÓN NO ENCONTRADA

Se comprueba el envío de un error en caso de que un UA solicite eliminar una suscripción que no existe.

Para comprobar este funcionamiento, un UA envía una operación "*Unsubscribe Service*" con en ID que no corresponde a ninguna suscripción. El DA deberá enviar una operación "*Error*" de tipo "*SUBSCRIPTION_NOT_FOUND"*.

El resultado es el esperado, al intentar borrar una suscripción inexistente, el servidor responde con un mensaje de error.

7.8. EXPIRACIÓN DE SUSCRIPCIÓN

Al registrar una suscripción, el UA recibe en la operación "*Subscribe Service Ack*" los valores de tiempo mínimo y máximo en los que debe actualizar la suscripción. En caso de que no se actualice en el tiempo indicado, la suscripción será eliminada.

Para comprobar que la suscripción se elimina correctamente, no se envía ninguna operación "*Update Subscription*".

El resultado el esperado, el DA borra las suscripciones que no se han actualizado en el tiempo adecuado.

7.9. ENVÍO DE NOTIFICACIÓN DE EVENTO

Se comprueba que el DA notifica de los cambios en los servicios a los agentes suscritos. Para la realización de esta prueba se siguen los siguientes pasos:

1. Suscripción de un UA a un tipo de servicio.
2. Registro de un servicio de ese tipo mediante el SA.
3. Comprobación de que el DA al recibir el registro, reenvía la información del servicio al UA.

El resultado es correcto, el DA envía notificaciones de los servicios suscritos.

7.10. RE-SUSCRIPCIÓN

En caso de que la base de suscripciones de un DA sea reiniciada, se producirán errores con los mensajes de actualización de las suscripciones. Esta prueba se realiza para comprobar que se puede reestablecer la suscripción correctamente.

El flujo de eventos sería el siguiente:

1. Subscripción de un UA a un tipo de servicio o UUID.
2. Reinicio de la base de suscripciones del DA.
3. El UA intenta actualizar la suscripción.
4. El DA responde con un error "*SUBSCRIPTION_NOT_FOUND*".
5. El UA registra de nuevo la suscripción en el DA.

En la primera suscripción, el ID de suscripción se obtiene de forma aleatoria. Sin embargo, para evitar realizar cambios en la base de suscripciones del UA por simplicidad para el cliente, la segunda suscripción no generará un nuevo ID, sino que reutilizará el original.

El resultado es correcto, el UA vuelve a crear la suscripción correctamente.

7.11. RESUMEN DE PRUEBAS

Tras realizar las pruebas, todas han resultado correctas. Por tanto, se concluye que los agentes son estables y robustos, cumpliendo el nivel de calidad esperado del proyecto. La siguiente tabla muestra un resumen de las pruebas realizadas.

ID	Prueba	Caso de uso	Resultado
1	Creación de suscripción - Funcionamiento normal	Creación de suscripción	Correcto
2	Creación de suscripción - Colisión de ID	Creación de suscripción errónea	Correcto
3	Creación de suscripción - Protocolo no soportado	Creación de suscripción errónea	Correcto
4	Actualización de suscripción - Funcionamiento normal	Actualización de suscripción	Correcto
5	Actualización de suscripción - Suscripción no encontrada	Actualización de suscripción errónea	Correcto
6	Eliminación de suscripción - Funcionamiento normal	Eliminación de suscripción	Correcto
7	Eliminación de suscripción - Suscripción no encontrada	Eliminación de suscripción errónea	Correcto
8	Envío de notificación de evento	Envío de notificación	Correcto
9	Expiración de suscripción	-	Correcto
10	Re-suscripción	-	Correcto

Tabla 15. Resumen de pruebas

Capítulo 8. PLANIFICACIÓN Y PRESUPUESTO

8.1. PLANIFICACIÓN TEMPORAL

Con el objetivo de emplear eficientemente el tiempo, antes de comenzar con el proyecto, se realizó un análisis de las distintas fases que deberían producirse durante la realización de este Trabajo Fin de Grado. Las fases de las que se compone este trabajo son:

- Inicio (5 semanas): Estudio a fondo de los drafts en los que se basa el desarrollo de XSSP, así como los trabajos de XSDF realizados anteriormente: XSLP y XSRP. Análisis del código inicial proporcionado para comprender el funcionamiento de las aplicaciones con exactitud. En esta fase también se estudia SLP por su similitud con XSDF.
- Planificación (3 semanas): Análisis de los casos de uso e identificación de los requisitos básicos de XSSP.
- Diseño (2 semanas): Realización de los diagramas de clases necesarios para decidir la funcionalidad y estructura de cada una de ellas.
- Desarrollo (3 semanas): Escritura del código necesario para la implementación de XSSP. Durante esta fase se realizan ligeras modificaciones con respecto a lo decidido en las fases de planificación y diseño.
- Pruebas (3 semanas): Para comprobar el correcto funcionamiento de las aplicaciones, se preparan baterías de pruebas para comprobar cada elemento individual, y finalmente se realizan pruebas de funcionamientos complejos para comprobar el funcionamiento de toda la arquitectura.
- Cierre (5 semanas): Realización de la memoria en base a las notas tomadas durante el desarrollo de las demás fases. Además, se realiza la presentación para la sesión de defensa.

Además, antes de comenzar la fase de inicio, se realizó un diagrama de Gantt con la duración estimada de cada fase. Debido a la complejidad del proyecto, se consideró una fecha de finalización anterior a la fecha de entrega del trabajo.

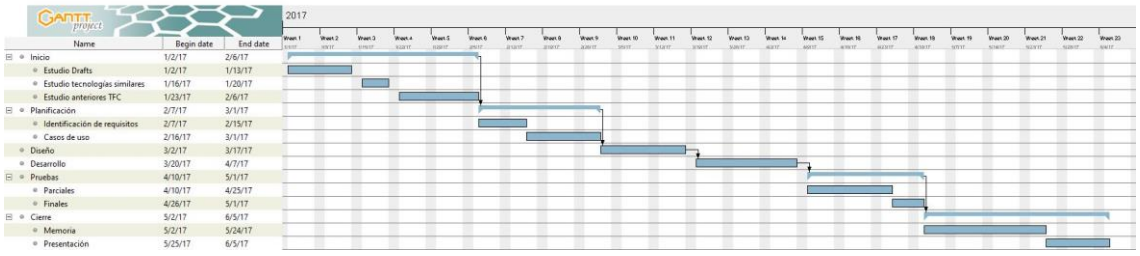

Ilustración 96. Planificación - Diagrama de Gantt

La planificación temporal ha sufrido retrasos importantes en la fase de desarrollo y en la elaboración de la memoria, por lo que la fecha de finalización ha sido 2 semanas después de la estimada inicialmente. Dado que se tuvo en cuenta que se pudiera dar esta situación, el retraso sufrido no afecta a la fecha de entrega del proyecto.

8.2. HERRAMIENTAS UTILIZADAS

Para el desarrollo del proyecto se ha optado por emplear herramientas gratuitas para reducir el coste de realización.

- CentOS 7 instalación mínima: Distribución de Linux basada en Red Hat empleada en las máquinas virtuales para ejecutar los distintos agentes de XSDF.
- Draw.io (online): Aplicación empleada para realizar los diagramas correspondientes a UPnP, Bonjour, SLP y XSDF.
- Gantt Project 2.8: Programa empleado para la realización del diagrama de Gantt ilustrado en la sección anterior.
- gedit 3.22: Editor de textos incluido en la distribución Sabayon, que se ha empleado para la manipulación del código java.
- Gimp 2.9: Aplicación empleada para retocar de ilustraciones generadas por las otras herramientas en caso de ser necesario.
- Java Development Kit 1.8.0: Conjunto de herramientas necesarias para la compilación y ejecución del código Java desarrollado.
- Microsoft Word 2016: Editor de textos empleado para la realización de la memoria. Se ha empleado la versión gratuita para universidades.
- Sabayon 16.11 edición con escritorio GNOME: Distribución de Linux basada en Gentoo instalada como sistema operativo principal en el ordenador empleado para el desarrollo del proyecto.
- StarUML: Herramienta empleada para la creación de diagramas de clases incluidos en el capítulo de Diseño.
- VirtualBox 5.1: Programa de virtualización empleado para la creación de las máquinas virtuales empleadas para ejecutar los distintos agentes de XSDF, y para la gestión de las mismas.

8.3. PRESUPUESTO

El presupuesto necesario para la realización de este proyecto puede dividirse en dos grupos: costes derivados del equipo necesario para el desarrollo y pruebas de las aplicaciones, y costes derivados de la mano de obra.

Debido a que todo el *software* utilizado es gratuito en el momento del desarrollo, no existen costes derivados del mismo. Tampoco existen costes relacionados con patentes o licencias de las especificaciones de XSDF.

Para el desarrollo, ha sido necesario un ordenador portátil que pudiera ejecutar simultáneamente y con fluidez una máquina virtual por cada agente. Se calcula que como máximo se ejecutarán 5 agentes a la vez y que cada máquina deberá disponer de 1 GB de memoria RAM. Dado que la instalación se realizará en versión de consola de comandos, la memoria gráfica no se considera una característica a tener en cuenta.

El equipo elegido pues es un HP NoteBook 15-AY509NS, con un coste de 450€ [21] y con un tiempo de vida aproximado de 5 años.

Coste total	Tiempo de vida	Amortización
450 €	5 años	90 €/año

Tabla 16. Costes de equipo

Por otra parte, para el desarrollo completo del proyecto, han sido necesarias dos personas: el alumno al que se considerará ingeniero *junior*, encargado del desarrollo principal, y el tutor, ingeniero *senior* encargado de la aprobación, supervisión y corrección del proyecto.

El número aproximado de horas dedicadas ha sido de 600 por parte del alumno, y 30 por parte del tutor.

Rol	Coste por hora	Horas	Coste total
Alumno	17 €/hora	600	10.200 €
Tutor	35 €/hora	30	1.050 €

Tabla 17. Costes de mano de obra

Los costes son un cálculo aproximado del salario bruto del empleado, más los costes indirectos que implica su actividad.

Los costes totales para el proyecto ascienden a 11.340 €

Concepto	Coste
Equipo	90 €
Mano de obra	11.250 €
TOTAL	11.340 €

Tabla 18. Presupuesto total

Capítulo 9. CONCLUSIONES Y TRABAJOS FUTUROS

9.1. CONCLUSIONES

Mediante este proyecto se buscaba el desarrollo de un protocolo que permitiera la gestión de suscripciones integrado en el entorno de descubrimiento de servicios XSDF. Los objetivos concretos se desarrollaron en el primer capítulo de este documento y son la implementación en Java de un cliente y un servidor XSSP que se integraran en la plataforma existente, y el desarrollo de documentación que ayudara a comprender el trabajo realizado.

Además, ha sido necesario mejorar la especificación del proyecto XSSP debido a problemas encontrados en la misma durante su implementación. Los principales cambios han sido los siguientes y son descritos con detalle en el apartado 4.4:

1. La inclusión de un identificador de la suscripción en todos los mensajes XSSP.
2. Se ha reemplazado la estructura original del elemento "*Subscribe Information*".
3. La notificación de cambios en los servicios a los que está suscrito un cliente, mediante el propio protocolo XSSP.

Tras el desarrollo de todos los puntos necesarios en esta memoria, se considera que todos los objetivos se han cumplido con éxito.

Con la adición de XSSP, la implementación de XSDF ofrece todas las funcionalidades definidas en la especificación menos la sincronización de los DAs en el momento de su conexión inicial, que será soportado tras la implementación de XSTP.

La mayor complejidad de este proyecto ha sido el estudio riguroso de las funcionalidades XSDF desarrolladas con anterioridad para posibilitar la coordinación de todos los agentes y para lograr un comportamiento similar entre los protocolos. La arquitectura de XSDF es mucho más compleja de lo que puede parecer en un primer momento.

Gracias a este proyecto se han adquirido habilidades sobre todo a la hora de emplear código desarrollado por diferentes personas con formas diferentes de programar. Además, se ha aprendido a realizar fases del proyecto que no se han realizado habitualmente durante el grado, como la elaboración de documentación o la creación de los diagramas de clases y secuencias expuestos en esta memoria. Por último, se ha puesto en práctica todo lo aprendido anteriormente de forma independiente, como podría suceder en un entorno laboral.

9.2. TRABAJOS FUTUROS

Con la finalización de este proyecto, tres de los cuatro protocolos que conforman XSDF han sido debidamente analizados e implementados.

Siguiendo el orden lógico, tras la conclusión de la implementación de XSSP, se debería realizar un Trabajo Fin de Grado que consistiera en la implementación del último protocolo de XSDF, el *eXtensible Service Transfer Protocol* o XSTP. Gracias a su implementación se lograría una de las mayores ventajas de XSDF: la capacidad de ser escalado a redes de gran tamaño mediante el despliegue de múltiples agentes de directorio sincronizados.

Adicionalmente, el último proyecto de implementación de XSDF deberá realizar un despliegue de la arquitectura completa para comprobar el buen funcionamiento de todos los protocolos y agentes.

Otra posibilidad sería el desarrollo de un proyecto de mejora del entorno XSDF para optimizar los recursos consumidos por los agentes haciéndolos más simples o modulares, o incrementar la eficiencia de la red mediante la creación de agentes intermedios que actúen como proxy o caché.

De forma paralela a la implementación en lenguaje Java se está realizando la implementación en lenguaje C que, a pesar de no estar en un estado tan avanzado, permitirá la ejecución de los agentes en sistemas sin soporte Java.

Dado el alcance de los protocolos que conforman XSDF, se podría realizar el desarrollo de los agentes para dispositivos menos tradicionales, como podrían ser dispositivos IoT, *tablets* o *smartphones*.

ANEXO I: ACRÓNIMOS

ACK: Acknowledgement

DA: Directory Agent

DHCP: Dynamic Host Configuration Protocol

DNS: Domain Name System

DNS-SD: DNS Service Discovery

HTTP: HyperText Transfer Protocol

IANA: Internet Assigned Numbers Authority

IETF: Internet Engineering Task Force

LAN: Local Area Network

mDNS: Multicast DNS

mDNS: Multicast DNS Service Discovery

NT: Notification Type

SA: Service Agent

SCTP: Stream Control Transmission Protocol

SDP: Service Discovery Protocol

SSDP: Simple Service Discovery Protocol

SLP: Service Location Protocol

TCP: Transmission Control Protocol

TLS: Transport Security Layer

TTL: Time To Live

UA: User Agent

UDA: UPnP Device Architecture

UDP: User Datagram Protocol

URL: Universal Resource Locator

USN: Unique Service Name

UPnP: Universal Plug and Play

VM: Virtual Machine

XBE: eXtensible Binary Encoding

XML: eXtensible Markup Language

XSDF: eXtensible Service Discovery Framework

XSLP: eXtensible Service Location Protocol

XSRP: eXtensible Service Registration Protocol

XSSP: eXtensible Service Subscription Protocol

XSTP: eXtensible Service Transfer Protocol

BIBLIOGRAFÍA

[1] UPnP Forum, "UPnP Device Architecture 2.0", 2015.

[2] Apple Computer, "Bonjour: Connect computers and electronic devices automatically", 2005.

[3] E. Guttman y C. Perkins, "Service Location Protocol, Version 2", IETF RFC2608, 1999.

[4] J. F. Martín, "Diseño e implementación Java del eXtensible Service Location Protocol (XSLP)", Trabajo Fin de Grado de la Universidad Carlos III de Madrid, 2008.

[5] I. Quiroga, "Diseño e implementación Java del eXtensible Service Registration Protocol (XSRP)", Trabajo Fin de Grado de la Universidad Carlos III de Madrid, 2009.

[6] M. Urueña y D. Larrabeiti, "eXtensible Service Subscription Protocol (XSSP) v00", IETF Internet Draft, 2004.

[7] M. Urueña y D. Larrabeiti, "Overview of the eXtensible Service Discovery Framework (XSDF) v00", IETF Internet Draft, 2004.

[8] M. Urueña y D. Larrabeiti, "eXtensible Service Discovery Framework (XSDF): Common Elements v00", IETF Internet Draft, 2004.

[9] M. Urueña y D. Larrabeiti, "eXtensible Binary Encoding (XBE32) v02", IETF Internet Draft, 2007.

[10] M. Urueña y D. Larrabeiti, "eXtensible Service Location Protocol (XSLP) v00", IETF Internet Draft, 2004.

[11] M. Urueña y D. Larrabeiti, "eXtensible Service Registration Protocol (XSRP) v00", IETF Internet Draft, 2004.

[12] M. Urueña y D. Larrabeiti, "eXtensible Service Transfer Protocol (XSTP) v00", IETF Internet Draft, 2004.

[13] Free Software Foundation, "GNU General Public License V3", 2007.

[14] Business Insider, "Global Internet Device Installed Base Forecast", 2014.

[15] J. Díaz Jiménez, "Protocolos de Descubrimiento de Servicios: Una mirada conceptual", Barranquilla: Universidad Simón Bolivar, 2010.

[16] E. Guttman, "Service Location Protocol: Automatic Discovery of IP Network Services", IEEE, 1999.

[17] E. Guttman y C. Perkins, "Service Templates and Service: Schemes", IETF RFC2609, 1999.

[18] P. Lei, "An Overview of Reliable Server Pooling Protocols", IETF RFC5351, 2008.

[19] P. Leach, "A Universally Unique IDentifier (UUID) URN Namespace", IETF RFC4122, 2005.

[20] M. Fernández, "Diseño e implementación del API del eXtensible Binary Encoding (XBE32)", Trabajo Fin de Grado de la Universidad Carlos III de Madrid, 2005.

[21] «HP NoteBook 15-AY509NS» PC Componentes, [En línea]. Available: https://www.pccomponentes.com/hp-notebook-15-ay509ns-intel-core-i3-6006u-8gb-500gb-156. [Último acceso: 10 06 2017].